Solving Large Linear Systems in an Implicit Thermohaline Ocean Model

Arie de Niet

This research was financially supported by the Dutch Technology Foundation STW (GWI.5798).

Rijks*universiteit* Groningen

Solving Large Linear Systems in an Implicit Thermohaline Ocean Model

Proefschrift

ter verkrijging van het doctoraat in de
Wiskunde en Natuurwetenschappen
aan de Rijks*universiteit* Groningen
op gezag van de
Rector Magnificus, dr. F. Zwarts,
in het openbaar te verdedigen op
vrijdag 22 juni 2007
om 13.15 uur

door

Arie Christiaan de Niet

geboren op 21 oktober 1977
te Groningen

Promotor:	Prof. dr. A. E. P. Veldman
Copromotor:	Dr. ir. F. W. Wubs
Beoordelingscommissie:	Prof. dr. ir. H. A. Dijkstra Prof. dr. ir. G. S. Stelling Prof. dr. H. A. van der Vorst

ISBN 90-367-3084-6

Contents

Chapter 1

Introduction

1.1 The challenge

In February 2007 the Intergovernmental Panel on Climate Change (IPCC) came out with a report [59] containing new scientific data on the effect of greenhouse gases on the earth's climate. Its main messages are that the global warming of the earth is undoubted, and that, with high probability, it is caused largely by human activity. Scientists predict that within a few decades the Arctic sea might be ice free during summers. The floating ice at the North Pole however is negligible compared to the enormous icecap on Greenland. The massive release of fresh water stored in this icecap can have dramatic impact on the earth's climate.

For example the mild climate in North-Western Europe is caused by the Gulf Stream that carries warm relatively salt water along the surface from the Gulf of Mexico across the North Atlantic to Europe. On its way to the north the flow cools down and sinks. It returns in southern direction along the bottom. There is an ongoing debate about the stability of the Gulf Stream. What happens if the flow is distorted by a bulk of fresh water from melting ice on Greenland? Could the Gulf Stream collapse? If the answer is yes, it would eventually weaken the warming due to the emission of green house gases in North-Western Europe.

The Gulf Stream is part of the so-called conveyor belt depicted in Figure 1.1: cooling down and sinking in the North Atlantic and upwelling and heating in the Pacific and Indian ocean. This global ocean circulation largely determines the earth's climate. It is therefore very important to study how it reacts to large perturbations. The main way to do this is to construct a computer model that mimics the physics of the global ocean circulation.

The global ocean circulation is a complex physical phenomenon. The flow is driven by (i) the wind, (ii) differences in temperature due to varying surface heating by the Sun, and (iii) differences in salinity due to evaporation (e.g. near the equator), precipitation, inflow from rivers and melting ice. The flow becomes even more complex due to the presence of the Coriolis force, which is caused by the rotation of the earth. Furthermore we have continents spread over the domain and a bottom topography with steep slopes.

We would like to compute this complex three dimensional flow. But more than that: we

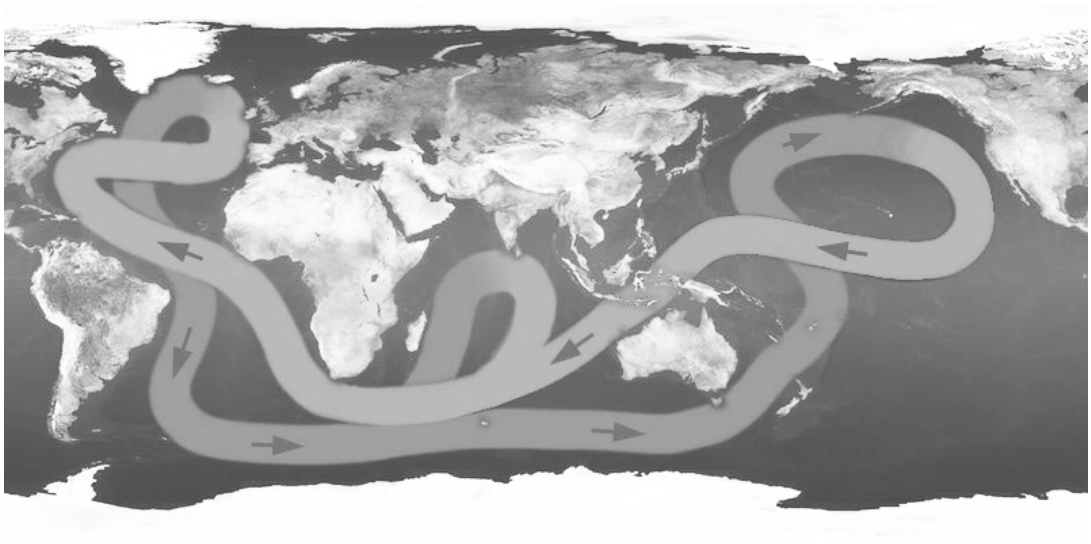


Figure 1.1: The conveyor belt of the thermohaline ocean circulation (from [29]).

would like to predict how the flow reacts to large perturbations.

At the Institute of Marine and Atmospheric research Utrecht (IMAU), the computer model THCM has been developed. The model computes three dimensional ocean flows, which involves the flow velocity in three directions, pressure, temperature and salinity. The relations between all these quantities are given by nonlinear partial differential equations and these have been approximated on a grid covering the ocean. The resolution of the grid immediately influences the accuracy of the simulation. With THCM one is able to compute the effect of perturbations.

We note that modern computers only allow to work with simplified models containing a number of parameters related to physics that cannot be respresented in the simplification. The values of these parameters are only known approximately, hence we also have to study the sensitivity of the flow to these parameters. We will describe the model and the mathematics involved in the variation of parameters (in which we include perturbations) and the computation of the flow in the next chapter.

At the start of the research project described in this thesis, THCM already existed, however several numerical problems limited the applicability of the model. A calculational run of the global ocean circulation required simply too much computer memory and time, which limited the resolution that can be used. Therefore we have to speed-up the numerics. *The challenge for this thesis is to develop new numerical techniques, such that it is possible to do large scale computations on steady states of the global ocean circulation with sufficient accuracy.*

More about the numerics later on in this introduction, first we will describe the background of the research project in more detail.

1.2 Reconstruction of the time-mean absolute velocity field of the global ocean circulation

The global ocean flow is time dependent. It has all kinds of periodicity, because the flow is influenced by the position of the earth with respect to the Moon and the Sun, which gives daily, monthly and seasonal variation. There are phenomena with an even longer period, like for example El Niño, which occurs approximately every 4 years. Nevertheless in the main part of this thesis we will consider the global ocean circulation as a steady flow, as happened in many studies in the past.

The aim of the research project is the reconstruction of the time-mean absolute velocity field of the global ocean circulation. To achieve this, we formulated a number of subgoals:

(1) improve the physics in the model,

(2) tune the model using observations,

(3) speed-up the numerics.

To start with (1), the model used primitive mixing schemes for salt and temperature. Advanced mixing schemes available in the literature needed to be incorporated in THCM.

Subgoal (2) requires more explanation. We want to use satellite data for the sea surface height (SSH) and the geoid to tune the model.

The geoid is an isosurface of the gravity field of the earth, i.e. on the geoid the gravitational acceleration is constant. If there would be no flow at all the sea surface would coincide with the geoid. The geoid is non trivial, because the earth is not a perfect sphere and the gravity field is influenced by the presence of ridges and trenches at the bottom of the ocean. The difference between SSH and geoid contains information about the flow at the top layer of the ocean.

As the differences are small (in the Gulf Stream the deviation is in the order of magnitude of one meter), we need accurate data for the SSH and the geoid. For the SSH there is accurate time-mean data available. The measurement of the gravity field is much more difficult. The errors of the available data (from the GRACE mission) are in the same order of magnitude as the SSH itself. So the current geoid data are worthless for our purpose. Fortunately the GOCE mission [21] will launch a satellite that measures the geoid up to an accuracy of a few centimeters. The launch is scheduled at the end of 2007.

Given an accurate geoid and the SSH data we can construct the flow at the surface. Via a technique called data-assimilation we can tune the ocean model, such that the model gives a surface flow that is close to the observations. This allows us to reduce the model errors.

Of the three subgoals we mentioned, (1) and (2) were carried out by Terwisscha van Scheltinga, a PhD student in the same project at the IMAU. He developed the advanced mixing schemes for salinity and temperature, which we will describe in Section 2.2.2 in the next chapter. The new mixing schemes strengthen the coupling between temperature and salinity, which makes the solution of the equations more difficult and subgoal (3) even more important.

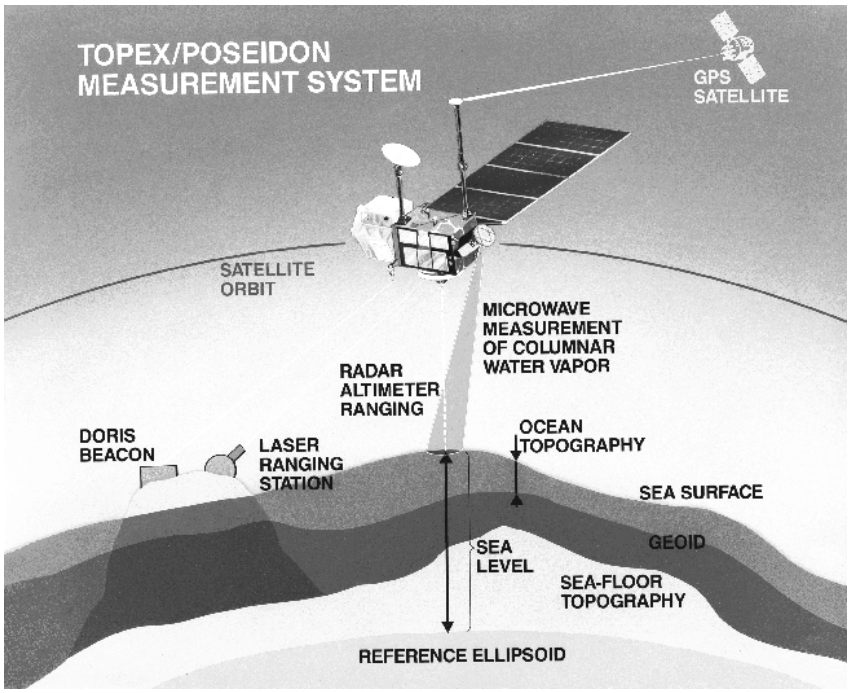


Figure 1.2: The sea surface height, the geoid and bottom topography.

Subgoal (2) required the development of a data-assimilation algorithm for the ocean flows [76]. The algorithm uses a number of successive model runs, which further emphasizes the importance of (3), the speed-up of the numerics.

1.3 Towards a new solver for the ocean model

The focus in this thesis will be on subgoal (3). The computation of thermohaline ocean flows and the variation of parameters therein require the solution of large linear systems involving the Jacobian matrix. The systems are hard to solve because of their size, which is in the order of a million unknowns, and the complexity of the underlying physics.

For many years the black-box method MRILU [10] (we will describe it in more detail in Chapter 3) was used to solve the large linear systems in THCM. Even though MRILU was able to solve the equations, where most alternative methods simply fail, it had serious drawbacks. The solution of the systems required much memory and computation time, which made it to the bottleneck of the model runs. Due to limitations in the linear system solver, it was impossible to increase the resolution, which is necessary for higher accuracy.

Hence we set the following target for this thesis: to build a new solver dedicated for THCM that

(3a) gives a considerable speed-up of the computations,

(3b) allows for parallelization,

(3c) combines with data-assimilation.

Condition (3b) is important, because if we have a solver than can be parallelized, we can run the model on one of the many available supercomputers, which not only have an enormous amount of memory, but also many processors which can work together to solve the problem in reasonable time.

We add condition (3c) because we want to use the solver and the ocean model in combination with data-assimilation. This gives some extra constraints on the solver. We will discuss them in the next chapter as well.

Of the three conditions the first is the most important. If we won't be able to satisfy condition (3a), the other two are of no interest anymore, which explains why the main part of the thesis is about fast system solvers.

In general there are two ways to solve large linear systems: via direct or via iterative methods. Direct methods try to compute a straightforward accurate solution of the equations, while the iterative methods construct a sequence of approximations for the solution until a certain desired accuracy has been reached. Direct methods usually require a huge amount of memory and computing time, while iterative methods are cheaper, but less robust in the sense that in many cases there is no guarantee that the method converges to the solution in reasonable time. Our systems are too large to apply direct solvers, but most iterative methods simply fail. In an overview paper on the history of iterative methods [65] it was suggested that improvements in the solution of complex partial differential equations, that are hard to solve by iterative methods, could come from a combination of both iterative and direct techniques. Hence the primary direction of search was the incorporation of direct techniques in MRILU.

In a large part of the thesis we will not study the ocean, but the saddle point problem, which also occurs in many other flows. This problem is somewhat simpler than the ocean problem, but likewise it is hard to solve by iterative methods. The detailed study of the solution of the saddle point problem helped in the design of a new solver for the ocean problem.

1.4 Outline of the thesis

The outline of the thesis is as follows. Chapter 2 describes the ocean model and the mathematics for parameter variations and determination of stability. In the final section of that chapter we will describe the systems that need to be solved and give some characteristics. The saddle point problem and its origin is described in Section 2.5. Chapter 3 is an introduction to the solution of large linear systems and the techniques involved in direct and iterative methods.

In the three chapters that follow, we study solution methods for more general large linear systems and the word "ocean" will be rare. In Chapter 4 we propose a new direct method for the solution of saddle point problems. We are able to prove that the method is numerically stable. It appears to perform better than other existing methods. In Chapter 5 we incorporate

techniques from direct methods in incomplete LU factorizations in the hope to be able to improve MRILU for coupled partial differential equations. We succeeded to combine both worlds; the results for Laplace equations are good, but for the saddle point problem they are disappointing. Chapter 6 deals with solvers for the saddle point problem, that exploit the structure in the matrix. We introduce two alternative solvers with are proven to have some nice mathematical properties. Numerical results show that these alternative solvers perform better than methods from literature.

Then we return to the solution of large linear systems in THCM. In Chapter 7 we propose a new solver for the systems. We will consider the performance of the new approach in Chapter 8 on a number of test cases, amongst which we have a global ocean circulation problem. The thesis finishes with Chapter 9 where we will summarize the results and draw conclusions.

Chapter 2

Building blocks of the thermohaline ocean circulation model

2.1 Introduction

The stability analysis for the global ocean circulation is performed with a computer program called THCM, which stems from *thermohaline circulation model*. The code is a product of the cooperation of a number of people at different institutes in the last few years and involves physical oceanography, dynamical system theory, numerical analysis and computer science. The work in this thesis is mainly in the field of numerical analysis, but in this chapter we will describe the building blocks of the model, including the blocks from the other fields.

We start with the equations that govern the ocean flow in Section 2.2. This is the most physics based part of the thesis. We treat both the analytic and discrete equations. Section 2.3 describes numerical continuation, a technique from dynamical system theory, that allows us to compute the effect of changes in parameters on the solution and perform linear stability analysis. This technique requires the Jacobian matrix of the discrete ocean equations. The construction of this matrix, its structure and some crucial properties are treated in Section 2.4. We pay attention to a subproblem of our system, that is the saddle point problem, which occurs also in many other flow equations.

2.2 The global ocean circulation

We give in a few lines the partial differential equations that describe the thermohaline ocean circulation. The word *thermohaline* means that the flow is driven by heat and salinity fluxes at the ocean-atmosphere interface. There are six quantities that play an important role: the flow velocity in three directions (u,v,w), the pressure (p), salinity (S) and the temperature (T). All variables are functions of space and time in a spherical coordinate system (ϕ,θ,z), where $z = r - r_0$, the deviation from the reference earth radius r_0.

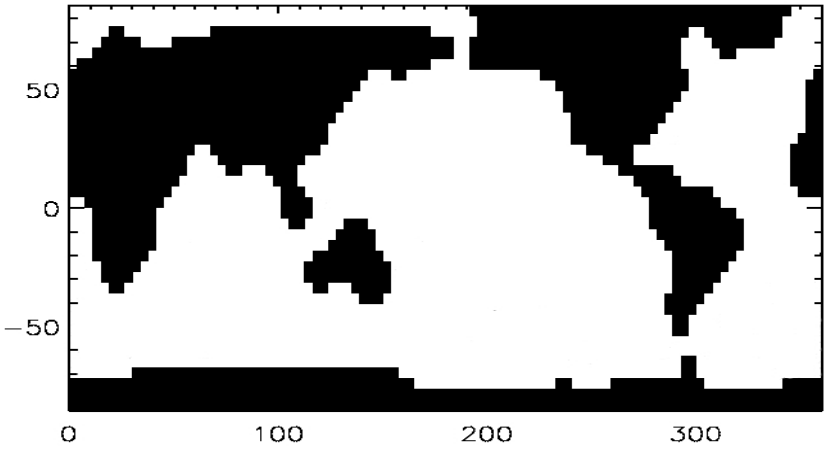


Figure 2.1: Top layer of the global grid with continents.

2.2.1 Model formulation

We consider flows in a spherical sector bounded by longitudes ϕ_w and ϕ_e ($-180 \leq \phi_e < \phi_w \leq 180$) and by latitudes θ_s and θ_n ($-90 < \theta_s < \theta_n < 90$). In case of the global ocean circulation we have $\phi_e = -180$ and $\phi_w = 180$ and the boundary conditions at the east and west boundary are periodic. Continents are allowed in the domain as well as a bottom topography (see Figure 2.1 for an example of the grid). So the ocean basin has a variable depth $D(\phi,\theta)$ and is bounded vertically by $z = -D(\phi,\theta)$ and a nondeformable ocean-atmosphere boundary $z = 0$. The flows in this domain are forced by a heat flux Q_H (in Wm^{-2}), a zonal wind stress field (τ^ϕ, τ^θ) (in Pa) and a virtual salt flux Q_S (in ms^{-1}), which includes evaporation and fresh water fluxes due to melting ice, rivers and precipitation. The heat flux Q_H is proportional to the temperature difference between the sea-surface temperature T and a prescribed atmospheric temperature T_S, i.e.,

$$Q_H = -\lambda_T (T - T_S) \tag{2.1}$$

where λ_T (in $\mathrm{Wm}^{-2}\mathrm{K}^{-1}$) is a constant exchange coefficient. The virtual salt flux Q_S is similarly formulated as a restoring condition and is given by

$$Q_S = -\lambda_S (S - S_S) \tag{2.2}$$

where λ_S (in ms^{-1}) is an exchange coefficient. Both wind and buoyancy forcing are distributed as a body forcing, implicitly defined by (2.1) and (2.2), over the first (upper) layer of the ocean having a thickness H_m.

Temperature and salinity differences in the ocean cause density differences according to

$$\rho = \rho_0 (1 - \alpha_T (T - T_0) + \alpha_S (S - S_0)) \tag{2.3}$$

where α_T and α_S are the volumetric expansion coefficients and T_0, S_0 and ρ_0 are reference quantities. We use the Boussinesq approximation, which means that density differences have

effect only in combination with g, the acceleration due to gravity. Density variations in the horizontal momentum and continuity equations can be ignored. Under this approximation the density is equal to the constant reference density ρ_0, except in the vertical momentum equation where we use (2.3). Furthermore we use the hydrostatic approximation, i.e. accelerations in the vertical are negligible, which reduces the vertical momentum equation to $\frac{\partial p}{\partial z} = -\rho g$. With r_0 and Ω being the radius and angular velocity of the Earth, the governing equations for the zonal (u), meridional (v) and vertical velocity (w) and the dynamic pressure p (the hydrostatic part has been subtracted) become

$$\begin{aligned}
\frac{Du}{dt} - uv\tan\theta - 2\Omega\, v\sin\theta + \frac{1}{\rho_0 r_0\cos\theta}\frac{\partial p}{\partial\phi} &= \\
A_V\frac{\partial^2 u}{\partial z^2} + A_H L_u(u,v) &+ \frac{\tau_0}{\rho_0 H_m}\tau^{\phi}\mathscr{G}(z) && \text{(2.4a)}\\
\frac{Dv}{dt} + u^2\tan\theta + 2\Omega\, u\sin\theta + \frac{1}{\rho_0 r_0}\frac{\partial p}{\partial\theta} &= \\
A_V\frac{\partial^2 v}{\partial z^2} + A_H L_v(u,v) &+ \frac{\tau_0}{\rho_0 H_m}\tau^{\theta}\mathscr{G}(z) && \text{(2.4b)}\\
\frac{\partial p}{\partial z} &= \rho_0 g(\alpha_T T - \alpha_S S) && \text{(2.4c)}\\
\frac{\partial w}{\partial z} + \frac{1}{r_0\cos\theta}\left(\frac{\partial u}{\partial\phi} + \frac{\partial(v\cos\theta)}{\partial\theta}\right) &= 0 && \text{(2.4d)}\\
\frac{DT}{dt} - R(\rho)T &= \frac{(T_S - T)}{\tau_T}\mathscr{G}(z) && \text{(2.4e)}\\
\frac{DS}{dt} - R(\rho)S &= \frac{(S_S - S)}{\tau_S}\mathscr{G}(z) && \text{(2.4f)}
\end{aligned}$$

where $\mathscr{G}(z) = \mathscr{H}(z/H_m + 1)$ and $\mathscr{H}$ is a continuous approximation of the Heaviside function. Furthermore, C_p is the constant heat capacity and $\tau_T = \rho_0 C_p H_m/\lambda_T$ and $\tau_S = H_m/\lambda_S$ are the surface adjustment time scales of heat and salt, respectively. In addition,

$$\begin{aligned}
\frac{D}{dt} &= \frac{\partial}{\partial t} + \frac{u}{r_0\cos\theta}\frac{\partial}{\partial\phi} + \frac{v}{r_0}\frac{\partial}{\partial\theta} + w\frac{\partial}{\partial z}\\
L_u(u,v) &= \nabla_H^2 u + \frac{u}{r_0^2\cos^2\theta} - \frac{2\sin\theta}{r_0^2\cos^2\theta}\frac{\partial v}{\partial\phi}\\
L_v(u,v) &= \nabla_H^2 v + \frac{v}{r_0^2\cos^2\theta} + \frac{2\sin\theta}{r_0^2\cos^2\theta}\frac{\partial u}{\partial\phi}\\
\nabla_H^2 &= \frac{1}{r_0^2\cos\theta}\left[\frac{\partial}{\partial\phi}\left(\frac{1}{\cos\theta}\frac{\partial}{\partial\phi}\right) + \frac{\partial}{\partial\theta}\left(\cos\theta\frac{\partial}{\partial\theta}\right)\right]
\end{aligned}$$

In the equations (2.4a-b), A_H and A_V are the horizontal and vertical momentum (eddy) viscosity which we will take constant. The operator $R(\rho)$ in (2.4e-f) represent the mixing of heat and salt and will be discussed in the next subsection.

2Ω	=	$1.4\cdot10^{-4}$	$[s^{-1}]$	r_0	=	$6.4\cdot10^{6}$	$[m]$
τ_T	=	$7.5\cdot10^{1}$	$[days]$	τ_S	=	$7.5\cdot10^{1}$	$[days]$
α_T	=	$1.0\cdot10^{-4}$	$[K^{-1}]$	α_S	=	$7.6\cdot10^{-4}$	$[-]$
C_p	=	$4.2\cdot10^{3}$	$[J(kgK)^{-1}]$	ρ_0	=	$1.0\cdot10^{3}$	$[kgm^{-3}]$
A_V	=	$1.0\cdot10^{-3}$	$[m^2s^{-1}]$	A_H	=	$1.0\cdot10^{4}$	$[m^2s^{-1}]$

Table 2.1: *Standard values of parameters used in the numerical calculations.*

Slip conditions are assumed at the bottom boundary, while at all lateral boundaries no-slip conditions are applied. At all lateral boundaries and the bottom boundary, the heat flux is zero. As the forcing is represented as a body force over the first layer, slip and no-flux conditions apply at the ocean surface. Hence, the boundary conditions are

$$z=-D,0 \quad : \qquad \frac{\partial u}{\partial z}=\frac{\partial v}{\partial z}=w=\frac{\partial T}{\partial z}=\frac{\partial S}{\partial z}=0, \tag{2.5a}$$

$$\phi=\phi_w,\phi_e \quad : \qquad u=v=w=\frac{\partial T}{\partial \phi}=\frac{\partial S}{\partial \phi}=0, \tag{2.5b}$$

$$\theta=\theta_s,\theta_n \quad : \qquad u=v=w=\frac{\partial T}{\partial \theta}=\frac{\partial S}{\partial \theta}=0. \tag{2.5c}$$

In case of periodic boundary conditions

$$\phi=\phi_w=180,\phi_e=-180: \quad u_w=u_e,v_w=v_e,w_w=w_e,T_w=T_e,S_w=S_e. \tag{2.6}$$

Parameters that are fixed in the calculations are the same as in typical large-scale low-resolution ocean general circulation models and their values are listed in Table 2.1. Other parameter values will be specified in the sections where we present the results of numerical experiments.

2.2.2 Tracer mixing representation

To avoid spurious diapycnal mixing (i.e. non-physical mixing in the direction of the density gradient) in the model of the previous section, in Equations (2.4f-g) so-called neutral physics fluxes have to be specified. Background and details of these fluxes and of the so-called small slope approximation used can be found in [36]. Expressions are summarized in section 14.3 of [36] and in our notation the mixing operator becomes

$$R(\rho)=\begin{bmatrix} \nabla_H^T & \frac{\partial}{\partial z} \end{bmatrix}\begin{bmatrix} K_H\mathbf{I} & \eta_M(K_H-\eta_G\kappa)\mathbf{S} \\ \eta_M(K_H+\eta_G\kappa)\mathbf{S}^T & \eta_M K_H\mathbf{S}^T\mathbf{S}+K_V \end{bmatrix}\begin{bmatrix} \nabla_H \\ \frac{\partial}{\partial z} \end{bmatrix} \tag{2.7}$$

where K_H is the neutral diffusivity and κ is the [31] skew diffusive mixing coefficient. In addition, the neutral slope vector $\mathbf{S}$ is given by

$$\mathbf{S}=\frac{\nabla_H\rho}{\frac{\partial\rho}{\partial z}} \tag{2.8}$$

with ρ as in (2.3).

Note that we have introduced two homotopy parameters η_M and η_G to be able to continue smoothly between constant horizontal diffusivity ($\eta_M = 0$) and neutral mixing ($\eta_M = 1$) and from no meso-scale eddy representation ($\eta_G = 0$) to the [31] representation ($\eta_G = 1$); the latter is referred below as GM-mixing.

We can only apply the representation above when the stratification is stable and when the slope of the isopycnals is small. To limit the slopes, we apply a similar procedure as in [15] by defining α_m as a maximum permissible slope angle and then multiplying η_M by the function $f(\frac{\partial\rho}{\partial z})$, where

$$f(x) = 0 \quad , x \leq -\xi \text{ or } x > 0 \tag{2.9a}$$

$$f(x) = 3(\frac{x+\xi}{\delta})^2 - 2(\frac{x+\xi}{\delta})^3 \quad , -\xi \leq x \leq -\xi + \delta \tag{2.9b}$$

$$f(x) = 1 \quad , -\xi + \delta \leq x \leq -\delta \tag{2.9c}$$

$$f(x) = 1 - 3(\frac{x+\delta}{\delta})^2 + 2(\frac{x+\delta}{\delta})^3 \quad , -\delta \leq x \leq 0 \tag{2.9d}$$

where

$$\xi = \frac{\mid \nabla_H \rho \mid}{\tan \alpha_m} \tag{2.10}$$

and $\delta = 0.05 \tan \alpha_m$. In this way, the mixing coefficients are smoothly tapered to zero in regions where the slope becomes too large or where the stratification is physically unstable.

The coefficient K_V in (2.7) is the vertical mixing coefficient. Whereas in many ocean-climate models, usually K_V is taken constant (or spatially prescribed), too high values (in the order of 10^{-4} m^{2}s^{-1}) were needed to obtain a realistic strength of the meridional overturning. Mixing in a stratified ocean, however, requires a transfer from kinetic to potential energy. The available energy for mixing, say e, is a topic of current research [88] and it is supplied by the wind, the tides and buoyancy forcing. An overview of the processes responsible for the spatial pattern of e is, for example, given in [44]. According to a production-dissipation equilibrium of the turbulent kinetic energy [77], the vertical mixing coefficient is defined as

$$K_V = K_V^0 + \frac{\Gamma_e}{N_b^2} \; ; \; N_b^2 = -\frac{g}{\rho_0}\frac{\partial \rho}{\partial z} \tag{2.11}$$

where $\Gamma_e = 0.2$ is the mixing efficiency and N_b is the buoyancy frequency. A background value K_V^0 can be attributed to internal wave breaking and it is taken to be constant.

In case of an unstable stratification, $N_b^2 < 0$, additional mixing occurs through convective overturning. We can take this mixing into account through an additional mixing coefficient $K_V^c \gg K_V^0$ by formulating K_V as

$$K_V = K_V^0 + \mathscr{F}(N_b^2)K_V^c + \eta_V \mathscr{F}(-N_b^2)\frac{\Gamma_e}{N_b^2} \tag{2.12}$$

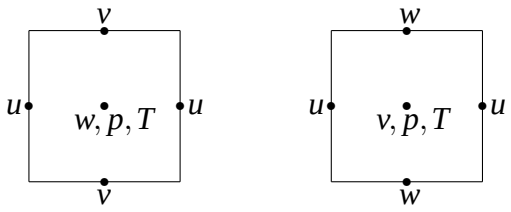


Figure 2.2: Positioning of variables in the horizontal C-grid, topview (left) and vertical cross section (right). The vertical layout is called the Lorenz grid.

where η_V is another homotopy parameter which can be used to study the situations with only constant mixing coefficient ($\eta_V = 0$) and variable mixing ($\eta_V = 1$). Furthermore, $\mathscr{F}$ is a mixing profile function which we take as

$$\mathscr{F}(x) = \max\{\tanh(-x^3), 0\} \tag{2.13}$$

such that additional convective mixing is generated smoothly as soon as $N^2 < 0$.

2.2.3 The discrete equations

The equations are discretized in space using a second order accurate control volume discretization method. The grid is stretched vertically, such that we have more cells close to the surface. The grid cells have indices $i = 1,...,N$, $j = 1,...,M$, $k = 1,...,L$.

In the horizontal direction we use a B-grid and in the vertical direction a C-grid. In atmospheric and oceanographic sciences the vertical layout, where the vertical velocity w is placed at the cell faces and the variables u, v, p, T, S on the plane halfway the cell, is called a *Lorenz grid*. In previous versions of the code (see [86]) a C-grid was used for both the horizontal and the vertical discretization. The Figures 2.2 and 2.3, show the horizontal C- and B-grid respectively. In [87] we motivated a choice for a B-grid instead of a C-grid via detailed Fourier analysis on a simplified ocean model with Cartesian grid and constant Coriolis force. The main reason is hat for small values of the horizontal viscosity A_H the C-grid gives components with high frequencies (wiggles) in the solution of the latitudinal velocity field, which do not appear using a B-grid.

The high frequency components at the C-grid are caused by the fact that the lateral Ekman layer cannot be resolved on a coarse mesh. The Ekman layer is a boundary layer that occurs at the eastern boundary of a basin. The thickness of the layer is typically

$$\delta_E = (A_H / f)^{1/2},$$

where $f = 2\Omega \sin\theta$. Note that both A_H and the Coriolis force are involved.

The Fourier analysis in [87] shows that in order to avoid wiggles in the solution we need a

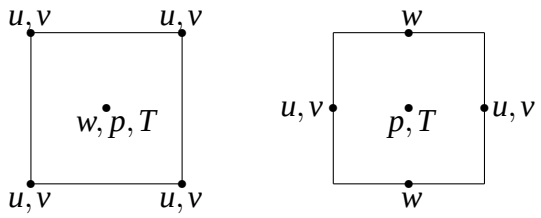


Figure 2.3: Positioning of variables in the horizontal B-grid, topview (left) and vertical cross section (right). The vertical layout is called the Lorenz grid.

longitudinal mesh size that satisfies

$$\Delta\phi_{\max} \ll \frac{2}{\pi^{1/3}} \left(\frac{\delta_E}{L} \right)^{2/3}.$$

Unfortunately in practice $\Delta\phi_{\max}$ often does not satisfy this bound and wiggles will occur in the solution on the C-grid. There are two ways to improve the results: decrease the mesh size which is computationally expensive, or increase A_H, which is unattractive from a physical point of view, because we would like to take A_H as small as possible. In case of the B-grid there is no such restriction.

In the equations (2.4a-2.4f) only derivatives of the pressure occur. The immediate consequence is that we can freely add a constant to the pressure field without changing the solution. The pressure is determined up to an additive constant. This holds as well for the discrete pressure field. On the C-grid we have a single degree of freedom, namely the constant vector for the pressure. However on the B-grid we have two degrees of freedom. There is a checker board-splitting of pressure nodes. Both the ”black” and the ”white” nodes form a vector that can be freely added to the solution of the pressure. In the next section we will see that we need the Jacobian of the discretized equations. Because of the degrees of freedom in the equations the Jacobian has two singular vectors in case of the B-grid and only one in case of the C-grid.

The spatially discretized model equations can be written in the form

$$\mathbf{M}\frac{d\mathbf{u}}{dt} = \mathbf{F}(\mathbf{u}) = \mathbf{L}(\mathbf{u}) + \mathbf{N}(\mathbf{u},\mathbf{u}) \tag{2.14}$$

where the vector $\mathbf{u}$ contains the unknowns (u, v, w, p, T, S) at each grid point and hence has dimension $d = 6 \times N \times M \times L$. The operators $\mathbf{M}$ and $\mathbf{L}$ are linear and $\mathbf{N}$ represents the nonlinear terms in the equations.

2.3 Numerical continuation

Steady state solutions lead to a set of nonlinear algebraic equations of the form

$$\mathbf{F}(\mathbf{u}, \mu) = 0. \tag{2.15}$$

Here the parameter dependence of the equations is made explicit through the parameter μ, which can be any of the parameters that we introduced in the previous section. Because μ is one dimensional, $\mathbf{F}$ is a nonlinear mapping from $\mathbb{R}^{d+1} \rightarrow \mathbb{R}^d$.

Given a solution for equations (2.15) we can vary the parameter μ and determine a branch of steady solutions. The computation of this branch is a *numerical continuation* problem. In this section we give a short introduction to the subject. For a more extended introduction we refer to [69, chapter 4] and [1].

By continuation we can find the relation between the parameter and the solution. It gives information about the effect of disturbances in the parameter, the stability of the solutions and whether multiple solutions exist for certain parameter values. On a branch we can encounter bifurcations, that is qualitative changes in the behavior of the solution, such as turning points, pitchfork and Hopf bifurcations.

Moreover continuation is useful as spin-up for complex problems with many problem parameters. If for some set of simple parameter values a solution is known, we can use this solution as a starting point for the continuation process. We follow the paths for all parameters until we reach the desired values.

In case of the ocean circulation problem, continuation of steady states is used for a combination of the reasons mentioned. The problem is huge and contains many problem parameters like earth rotation speed, the atmospheric temperature function, the precipitation, the wind field, the shape of the continents, the shape of the bottom of the ocean and many physical constants.

2.3.1 Pseudo-arclength parameterization

Because the equation $\mathbf{F}(\mathbf{u},\mu) = 0$ is underdetermined, we need an extra equation if we want to determine a particular solution on the branch. This equation is given by the parameterization of the curve, that defines a relation between $\mathbf{u},\mu$ and the parameterization variable s. We use the pseudo-arclength method [47], because it is able to handle turning points.

The pseudo-arclength parameter s of the curve $(\mathbf{u}(s),\mu(s))$ is defined implicitly by

$$\zeta \left\| \frac{d\mathbf{u}}{ds} \right\|_2^2 + \left(\frac{d\mu}{ds} \right)^2 = 1, \tag{2.16}$$

where the number ζ is used to place different emphasis on $\mathbf{u}$ and μ. We choose $\zeta = 1/d$, the inverse of the dimension of the vector $\mathbf{u}$. Given a solution $(\mathbf{u}(s_0),\mu(s_0)) = (\mathbf{u}_0,\mu_0)$, which can be an analytically known starting solution or a previously computed point on the branch, we can replace the derivatives by a first order approximation. Multiplication with the square of the step length $\Delta s = (s - s_0)$ gives the desired extra equation

$$q(\mathbf{u},\mu,s) = \zeta(\mathbf{u}-\mathbf{u}_0)^T(\mathbf{u}-\mathbf{u}_0) + (\mu-\mu_0)^2 - (s-s_0)^2 = 0. \tag{2.17}$$

Fixing s we want to solve the following equation

$$\begin{bmatrix} \mathbf{F}(\mathbf{u},\mu) \\ q(\mathbf{u},\mu,s) \end{bmatrix} = 0. \tag{2.18}$$

2.3.2 Predictor-Corrector method

Equation (2.18) is solved with a predictor-corrector method. The most commonly used predictor-corrector method is Euler-Newton continuation. The first-order accurate Euler method is used to predict the solution on the branch at s and that prediction is the starting point for a Newton iteration that computes a more accurate solution. The Newton method requires the $(d+1) \times (d+1)$ Jacobian matrix $\mathbf{J}_s(\mathbf{u},\mu)$ of (2.18) that is given by

$$\mathbf{J}_s(\mathbf{u},\mu) = \begin{bmatrix} \Phi & \mathbf{F}_\mu \\ q_\mathbf{u} & q_\mu \end{bmatrix}, \tag{2.19}$$

where Φ is the matrix of derivatives of $\mathbf{F}$ to $\mathbf{u}$, $\mathbf{F}_\mu$ the derivative to the parameter μ, $q_\mathbf{u}$ the derivative of q with respect to $\mathbf{u}$ and q_μ the derivative of q with respect to μ. Note that all the subblocks of the Jacobian matrix depend on $\mathbf{u}$ and μ.

The Euler predictor

$$(\mathbf{u}^{(0)},\mu^{(0)}) = (\mathbf{u}_0,\mu_0) + (\dot{\mathbf{u}}_0,\dot{\mu}_0)\Delta s, \tag{2.20}$$

requires the tangent $(\dot{\mathbf{u}}_0,\dot{\mu}_0)$, which is the derivative of $(\mathbf{u},\mu)$ with respect to s at $s = s_0$. We can compute the tangent in the following way. Differentiation of $\mathbf{F}(\mathbf{u},\mu) = 0$ with respect to μ gives

$$\Phi\frac{d\mathbf{u}}{d\mu} + \mathbf{F}_\mu = 0. \tag{2.21}$$

We can solve $d\mathbf{u}/d\mu$ from this equation. Furthermore we can substitute

$$\frac{d\mathbf{u}}{d\mu} = \frac{d\mathbf{u}}{ds}\frac{ds}{d\mu} \tag{2.22}$$

in Equation (2.16). After some rearrangement we get

$$\frac{d\mu}{ds} = 1/\sqrt{1+\zeta\left\|\frac{d\mathbf{u}}{d\mu}\right\|^2}. \tag{2.23}$$

To compute the tangent we solve Equation (2.21), compute $d\mu/ds$ with (2.23) and finally compute $d\mathbf{u}/ds$ with (2.22).

If we have two previously computed solutions, we can easily apply the secant method. which is favourable because it does not require the solution of a system with Φ. Say we have a point $(\mathbf{u}_{-1},\mu_{-1})$ on the branch before $(\mathbf{u}_0,\mu_0)$, then the prediction becomes

$$(\mathbf{u}^{(0)},\mu^{(0)}) = (\mathbf{u}_0,\mu_0) + \frac{(\mathbf{u}_0,\mu_0)-(\mathbf{u}_{-1},\mu_{-1})}{s_0 - s_{-1}}\Delta s. \tag{2.24}$$

Both the secant and Euler predictor are first order accurate in Δs, therefor it does not make a big difference which one we choose. At the starting point of the continuation there is no previously computed point, so there we are forced to use the Euler predictor.

The Newton iteration requires solution of the updates in the linear equation

$$\mathbf{J}_s\left(\mathbf{u}^{(k)},\mu^{(k)}\right)\begin{bmatrix}\Delta\mathbf{u}^{(k)}\\ \Delta\mu^{(k)}\end{bmatrix}=\begin{bmatrix}-\mathbf{F}\\ -q\end{bmatrix}, \tag{2.25}$$

where the right hand side is a function of $(\mathbf{u}^{(k)},\mu^{(k)})$. The next approximation is

$$(\mathbf{u}^{(k+1)},\mu^{(k+1)})=(\mathbf{u}^{(k)},\mu^{(k)})+(\Delta\mathbf{u}^{(k)},\Delta\mu^{(k)}). \tag{2.26}$$

The iteration stops as soon as $\|(\Delta\mathbf{u}^{(k)},\Delta\mu^{(k)})\|_\infty<\varepsilon$, usually ε is 10^{-3}.

In each iteration of the Newton process we have to solve Equation (2.25) where $\mathbf{J}_s$ is given by (2.19). If we assume Φ is invertible we can multiply the first rows with Φ^{-1} which results in the following system

$$\begin{bmatrix}\mathbf{I} & \Phi^{-1}\mathbf{F}_\mu\\ q_\mathbf{u} & q_\mu\end{bmatrix}\begin{bmatrix}\Delta\mathbf{u}\\ \Delta\mu\end{bmatrix}=\begin{bmatrix}-\Phi^{-1}\mathbf{F}\\ -q\end{bmatrix}. \tag{2.27}$$

If we now introduce $\mathbf{v}$ and $\mathbf{w}$, the solutions of the equations

$$\Phi\mathbf{v}=-\mathbf{F},\quad\text{and}\quad\Phi\mathbf{w}=\mathbf{F}_\mu, \tag{2.28}$$

we can easily compute the updates via

$$\Delta\mu = -(q+q_\mathbf{u}\mathbf{v})/(q_\mu-q_\mathbf{u}\mathbf{w}), \tag{2.29}$$
$$\Delta\mathbf{u} = \mathbf{v}-\Delta\mu\mathbf{w}. \tag{2.30}$$

Summarizing, each step in the Newton iteration starts with the computation of $\mathbf{v}$ and $\mathbf{w}$ by solving (2.28). Then $\Delta\mu$ and Δu can be computed with equations (2.29-2.30). In this process the solution of the two systems with Φ is by far the most expensive.

Note that the second equation in (2.28) is equal to (2.21), so $\mathbf{w}=-d\mathbf{u}/d\mu$. In fact we get the Euler predictor for the next step for free. So we can use both the Euler-predictor and the secant predictor.

2.3.3 The adaptive Shamanskii corrector

The Newton method requires at each step the solution of system (2.25). To be able to solve the system we first of all have to construct it. We have to build the Jacobian $\mathbf{J}_s(\mathbf{u},\mu)$, or equivalently $\Phi(\mathbf{u},\mu)$ and the vector $\mathbf{F}_\mu(\mathbf{u},\mu)$. Because of the nonlinear terms in the equations and the complex mixing scheme, the computation of Φ is quite expensive. Given Φ and $\mathbf{F}_\mu$, we solve the systems (2.28) via a Krylov subspace iteration that demands the construction of a preconditioner for Φ. In Section 3.2 of the next chapter we will explain what we mean by Krylov subspace iteration and preconditioning. For now it is enough to know that the construction of a good preconditioner is the most expensive step in the procedure.

Instead of the Newton method we can use an alternative fixed point method. There is a wide range of such methods available, for an extended overview see [11]. We will shortly mention a few fixed point methods that are variants of Newton's method.

If we simply fix the Jacobian after the first step we get the *Newton-chord* method. The convergence rate will be linear instead of quadratic. If the first guess is relative close to the solution the convergence is acceptable. In general we will have to take smaller steps in the continuation method. The advantage of the Newton-chord method is that we avoid a lot of computations. The Jacobian Φ, the vector $\mathbf{F}_\mu$ and the preconditioner need to be computed only once. Because $\mathbf{F}_\mu$ is fixed, we have to solve only the first equation in (2.28) at each step of the iteration.

The *Shamanskii* method [70] combines the Newton-chord with the Newton method. The Jacobian is fixed for an a priori chosen number of steps. For example each third step will be an expensive Newton step and the two steps in between are much cheaper Newton-chord steps.

The last method can be improved to the *adaptive Shamanskii* method. Instead of choosing the number of Newton-chord steps on beforehand, one can decide to switch from Newton to Newton-chord based on the convergence of the fixed point iteration. The idea is simple: if the Newton-chord method is converging not fast enough, a Newton step is performed. However we have to determine when it is advantageous to take a single expensive Newton step instead of a number of cheap Newton-chord steps. Therefore we introduce T_n, the time needed for a Newton step, and T_{nc}, the time needed for an Newton-chord step. Further let κ_n be the convergence constant of the Newton method and κ_{nc} the convergence constant of the Newton-chord method. Hence if ε_k is the error of the k-th approximation of $(\mathbf{u},\mu)$ (in the Euclidean norm), then with the Newton method the next approximation will have an error close to $\varepsilon_{k+1} = \kappa_n \varepsilon_k^2$ and with the Newton-chord method $\varepsilon_{k+1} = \kappa_{nc}\varepsilon_k$. After l Newton-chord iterations the estimate of the error is $\varepsilon_{k+l} = (\kappa_{nc})^l \varepsilon_k$. From these expressions we can easily compute the number of Newton-chord iterations, that we need, to obtain the same accuracy as obtained by one step of the Newton method or, if $\kappa_n \varepsilon_k^2 < \varepsilon$, the target accuracy ε:

$$l = \log(\max(\kappa_n \varepsilon_k, \varepsilon/\varepsilon_k))/\log(\kappa_{nc}).$$

The amount of time needed by l inner iterations is $l \cdot T_{nc}$. As long as $l \cdot T_{nc} < T_n$, the time for the Newton-chord steps is less than the time for one Newton step, so it is beneficial to perform the Newton-chord steps. The relation $l \cdot T_{nc} < T_n$ can be rewritten such that we get

$$\kappa_{nc} < \exp\left[\log\left(\max\left(\kappa_n \varepsilon_k, \frac{\varepsilon}{\varepsilon_k}\right)\right) \cdot \frac{T_{nc}}{T_n}\right]. \tag{2.31}$$

After each inner iteration we can calculate new estimates for κ_{nc} and T_{nc} and check if this relation still holds. If it doesn't we perform a Newton step that gives new estimates for κ_n and T_n.

The adaptive Shamanskii method can be implemented easily and gives asymptotically the fastest convergence in time possible, therefore it is the corrector method of choice. In [86] we showed that one can save much time using the adaptive Shamanskii method instead of Newton's method.

2.3.4 Step size control

The last issue we want to address about continuation is the choice of $\Delta s = s - s_0$ in (2.17), i.e. the size of the step along the branch. We can simply fix the step size over a number of steps, but that is not very efficient. In practice we are mainly interested in the point(s) on the curve where the parameter equals some target value $\hat{\mu}$, for example a bifurcation point or the point where the parameter takes its physical values. We would like to step as fast as possible through the branches until we reach the desired value. We don't want to take unnecessarily small steps, but if we choose the step size too large we risk slow convergence in the correction process or even divergence from the branch. The step size has to be chosen such that on one hand the step size in the predictor is as large as possible, while on the other hand the number of iterations in the corrector is as small as possible. These demands are conflicting, so we have to find the balance between both. In this paragraph we describe a method that searches this balance.

Seydel [69, paragraph 4.6] suggests that there exists something like an *optimal number of iterations*, call it N_{opt} for the Newton iteration. Let N be the number of iterations in the last Newton process, if $N > N_{opt}$ we could better take a smaller step size and if $N < N_{opt}$ we may be able to take a larger one. This leads to the following definition for ξ, the multiplication factor for the step size,

$$\xi = \begin{cases} 0.5 & \text{if} \quad \frac{N_{opt}}{N} < 0.5 \\ \frac{N_{opt}}{N} & \text{if} \quad 0.5 \leq \frac{N_{opt}}{N} \leq 2 \\ 2 & \text{if} \quad 2 < \frac{N_{opt}}{N} \end{cases} ,$$

where the next step is $\xi \Delta s$. The increase or decrease is limited by a factor 2 in order to prevent too rapid changes in the size of the step. The method is not very sensitive to the precise value of N_{opt}, in case of the Newton method 5 seems a good value, in case of the adaptive Shamanskii the value 7 works fine.

The adaptive step size is combined with a *trial and error* method: if the corrector diverges or does not converge within $2N_{opt}$ iterations, the step is rejected and a new step is taken from the last solution with half the step size. The adaptive step size was successfully used in the computations in [86].

2.4 The Jacobian matrix

The solution of the two equations involving the Jacobian matrix Φ in (2.28) is the bottle neck in the computations. Hence, if we want to speed up THCM, we have to find a better way to solve these equations. Before we pay attention to the solution of large linear systems, we have to know more about construction and structure of the Jacobian.

2.4.1 Construction

In previous formulations of THCM [86], the entries in the Jacobian matrix were computed analytically and directly from the discretized equations. As the mixing formulations (2.7) and (2.12) are quite complicated, it turned out to be advantageous to use an additional numerical evaluation of the Jacobian matrix. In [14], the problem of how to estimate the Jacobian matrix **J** of a general mapping $\mathbf{F} : \mathbf{R^n} \rightarrow \mathbf{R^m}$ using the least number of function evaluations is discussed. The j-th column of **J** is approximated by

$$\mathbf{J}_j = \frac{\mathbf{F}(\cdots, \mathbf{x}_j + \varepsilon, \cdots) - \mathbf{F}(\cdots, \mathbf{x}_j, \cdots)}{\varepsilon}. \tag{2.32}$$

for small ε. In general, we need to calculate m of such differences to approximate the Jacobian. For sparse Jacobian matrices, a group of columns can be determined such that no two columns in this group have a nonzero element in the same row position. This is because, for large-scale problems, the elements of **F** are dependent on a limited number of variables x_j. Let C be such a group of columns and let $\mathbf{d} \in \mathbf{R}^m$ be a vector with $d_j = \varepsilon$ if $\mathbf{J}_j \in C$ and $d_j = 0$ otherwise. Then the difference

$$\mathbf{J}_C = \frac{\mathbf{F}(\mathbf{x} + \mathbf{d}) - \mathbf{F}(\mathbf{x})}{\varepsilon} \tag{2.33}$$

contains the non-zero elements of the columns belonging to C. By partitioning the columns of the Jacobian matrix in this way, we can more efficiently evaluate the Jacobian since the number of differences we need to calculate are strongly reduced. For the implementation of this approach, we used subroutines provided by [14]. These subroutines consists of (i) routines that determine the groups C, such that the number of groups is minimal; and (ii) routines that determine for a given group C the vector **d** and extract the columns $\mathbf{J}_j$ from $\mathbf{J}_C$. These routines require the sparsity pattern of the Jacobian matrix, i.e., the variables on which **F** depends, and a routine that evaluates $\mathbf{F}(\mathbf{x})$.

This method is applied only to the mixing formulations in (2.7) and (2.12). The rest of the entries in the Jacobian is computed analytically from the discrete equations.

The continuous formulation of convective adjustment in (2.12) combined with the numerical evaluation of the Jacobian, can cause large off-diagonal couplings between T and S in the Jacobian, which makes the solution of the equations more difficult.

2.4.2 Structure

In each step of the Newton process we have to solve one or two linear systems of equations with the matrix Φ in (2.19), which is by far the most expensive part of the computation.

The equation $\Phi\mathbf{u} = \mathbf{b}$ has the following structure

$$\begin{bmatrix} \mathbf{A}_{uv} & \mathbf{E}_{uv} & \mathbf{G}_{uv} & 0 \\ 0 & 0 & \mathbf{G}_w & \mathbf{B}_{TS} \\ \mathbf{D}_{uv} & \mathbf{D}_w & 0 & 0 \\ \mathbf{B}_{uv} & \mathbf{B}_w & 0 & \mathbf{A}_{TS} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{uv} \\ \mathbf{u}_w \\ \mathbf{u}_p \\ \mathbf{u}_{TS} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{uv} \\ \mathbf{b}_w \\ \mathbf{b}_p \\ \mathbf{b}_{TS} \end{bmatrix}, \tag{2.34}$$

The four rows in (2.34) represent the discrete version of the equations for conservation of momentum in longitudinal and latitudinal direction (equations (2.4a,2.4b)), hydrostatic equilibrium (equation (2.4c)), conservation of mass (equation (2.4d)) and conservation of heat and salt (equations (2.4e,2.4f)), respectively.

Each $\mathbf{G}_*$ represents the discrete gradient operator and each $\mathbf{D}_*$ the discrete divergence operator. Due to the spherical coordinate system these discrete operators are not necessarily each others transpose. However there exist diagonal matrices Λ_1 and Λ_2 such that $\mathbf{G}_*^T = \Lambda_1 \mathbf{D}_* \Lambda_2$, which is important for numerical stability. The matrices $\mathbf{A}_{TS}$ and $\mathbf{A}_{uv}$ are of convection-diffusion type, and $\mathbf{A}_{uv}$ includes the Coriolis terms. The matrices $\mathbf{B}_*$ represent the couplings between the dynamic and thermodynamic variables. The matrix $\mathbf{E}_{uv}$ represents the coupling to w in the material derivative in equations (2.4a,2.4b). This coupling is very weak hence we will ignore the block in this section. It is treated as another zero block which simplifies the calculations. As one can see, next to $\mathbf{E}_{uv}$ there are six zero blocks, which means that severable variables are not coupled. All the submatrices are structured and sparse and there are only a few nonzeros per row in each matrix.

We use clustering for u, v, because the equations (2.4a) and (2.4b) have the same structure. As the same holds for (2.4e) and (2.4f), also the variables T, S are clustered.

Because of the B-grid (see Figure 2.3), in principle there are not equally many unknowns of each kind. We added dummy variables to make sure there are precisely $d = 6 \times N \times M \times L$ unknowns. These dummy variables are located outside the domain and are described by trivial equations. In general, the dummy equations for w and p will generate nonzero diagonal entries in the second and third diagonal block of (2.34). To obtain the given structure, we have to remove these equations in advance. Immediate consequence is that the dimensions of the vectors $\mathbf{u}_*$ in (2.34) are no longer the same. Let d_* be the dimension of each $\mathbf{u}_*$, then it holds that $d_{uv} = d_{TS} = 2 \times N \times M \times L$ and $d_w < d_p \leq N \times M \times L$. The inequality $d_w < d_p$ holds because we have a Lorenz grid in the vertical direction (see Figure 2.3). Per water column, there is one internal pressure node more than there are vertical velocity nodes.

The matrix in (2.34) is huge, far from diagonal dominant and indefinite, i.e. it has eigenvalues with positive as well as negative real part. This makes the system hard to solve for both the direct and iterative methods, that we will describe in Chapter 3.

2.5 The saddle point problem

In the previous chapter we already mentioned the bottle neck in the computation of ocean flows: the solution of systems with the Jacobian matrix. This is by far the most expensive part. A step to models with a higher resolution is only possible if we improve the way the systems are solved. The problems in solving the Jacobian matrix of the ocean equations are largely caused by the fact that it stems from a set of coupled partial differential equations, which gives a complex matrix structure as in Equation (2.34). Only in the next chapter we will introduce

the reader to the solution of large systems of equations. First we want to describe a somewhat easier problem that partially resembles the structure of ocean matrix: the *saddle point problem*.

A saddle point problem occurs in several flow problems and is, alike the ocean system, hard to solve. In our quest for better solvers for the Jacobian in ocean flows, we studied the saddle point problem, because better insight in the solvers for the saddle point problems could help us in the design of a new solver for the ocean systems.

A saddle point problem is given by an equation

$$\mathbf{K}\mathbf{x} = \mathbf{b}, \tag{2.35}$$

where $\mathbf{K} \in \mathbb{R}^{(n+m)\times(n+m)}$ $(n \geq m)$ is a matrix of the special form

$$\mathbf{K} = \begin{pmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{C} \end{pmatrix}, \tag{2.36}$$

where $\mathbf{C} \in \mathbb{R}^{m\times m}$ is symmetric and negative semidefinite, $\mathbf{A} \in \mathbb{R}^{n\times n}$, and $\mathbf{B} \in \mathbb{R}^{n\times m}$. In this thesis we only consider $\mathbf{C} = \mathbf{0}$. Often the matrix $\mathbf{A}$ will be symmetric positive definite. Then the matrix $\mathbf{K}$ itself is symmetric indefinite. Like Φ in (2.34), $\mathbf{K}$ is indefinite, structured and sparse. A saddle point problem occurs in the numerical solution of partial differential equations with constraints, like for example the (Navier)-Stokes or Oseen equations.

2.5.1 The Navier-Stokes equations

The incompressible Navier-Stokes equations on an open bounded domain are

$$\begin{aligned} -\nu\Delta u + (u\cdot\nabla)u + \nabla p &= 0, \\ \nabla\cdot u &= 0. \end{aligned} \tag{2.37}$$

The linearized version of this equation is

$$\begin{aligned} -\nu\Delta u + (\bar{u}\cdot\nabla)u + \nabla p &= 0 \\ \nabla\cdot u &= 0, \end{aligned} \quad , \tag{2.38}$$

where the "wind" $\bar{u}$ is such that $\nabla\cdot\bar{u} = 0$. These *Oseen equations* occur if we solve the Navier-Stokes equations via a Picard iteration, where we take $u = u^{(m)}$ and $\bar{u} = u^{(m-1)}$, the previous guess for the solution. Discretization of the Oseen equations gives a saddle point problem with non-symmetric positive definite $\mathbf{A}$.

If we further simplify the equations and drop the convective term $((\bar{w}\cdot\nabla)u)$ we get the Stokes equation

$$\begin{aligned} -\nu\Delta u + \nabla p &= 0, \\ \nabla\cdot u &= 0. \end{aligned} \tag{2.39}$$

After discretization we get a saddle point problem with symmetric and positive definite $\mathbf{A}$.

2.5.2 Saddle point problems and ocean flows

There is an important connection between saddle point problems and the ocean problem. The Jacobian matrix Φ from (2.34) can be written as two coupled saddle point problems. This is easily shown if we apply the permutation matrix $\mathbf{Q}$, which corresponds to the block-permutation $(1,3,4,2)$, to Φ:

$$\mathbf{Q}\Phi\mathbf{Q}^T = \left[\begin{array}{cc|cc} \mathbf{A}_{uv} & \mathbf{G}_{uv} & 0 & \mathbf{E}_{uv} \\ \mathbf{D}_{uv} & 0 & 0 & \mathbf{D}_w \\ \hline \mathbf{B}_{uv} & 0 & \mathbf{A}_{TS} & \mathbf{B}_w \\ 0 & \mathbf{G}_w & \mathbf{B}_{TS} & 0 \end{array}\right]. \tag{2.40}$$

One immediately recognizes the two saddle point problems: the first is a Navier-Stokes system including Coriolis for u,v and p (the dynamical part) and the second is a saddle point problem for T,S and w (the thermodynamical part). The last system is certainly not symmetric, because $\mathbf{B}_{TS}$ is the discretization of the term $\rho_0 g(\alpha_T T - \alpha_S S)$ in (2.4c), so it is a constant matrix, whereas $\mathbf{B}_w$ is the discretization of $w\frac{\partial T}{\partial z}$ and $w\frac{\partial S}{\partial z}$ in (2.4e) and (2.4f) respectively, so it depends on T and S, and consequently we have $\mathbf{B}_{TS} \neq \mathbf{B}_w^T$.

Because Φ and $\mathbf{K}$ have similar properties we study the numerical solution of system (2.35) hoping to be able to apply similar techniques to Φ. Therefore in Chapters 4 and 6 the focus is on the saddle point problem and in Chapter 5 it plays an important role as well. The knowledge on the saddle point problem *is* relevant for the ocean system, as we will see in Chapter 7, where saddle point problems occur in the solution of equations with Φ.

2.6 Time stepping, data assimilation and stability analysis

So far we assumed that we solve steady states of the equations (2.4a-f), hence all the time derivatives vanished. However THCM can do time stepping as well. In fact, the algorithms for continuation of steady states and time integration have a lot in common. In this section we shortly pay attention to the solution of time dependent ocean problems, because the data assimilation algorithm uses time integration.

To solve the time dependent equation (2.14) we apply the theta-method

$$-\mathbf{M}\,\frac{\mathbf{u}^{(n+1)} - \mathbf{u}^{(n)}}{\Delta t} = \Theta\mathbf{F}\left(\mathbf{u}^{(n+1)}\right) + (1-\Theta)\mathbf{F}\left(\mathbf{u}^{(n)}\right) \tag{2.41}$$

with initial condition $\mathbf{u}^{(0)}$. The matrix $\mathbf{M}$ is given by

$$\mathbf{M} = \left[\begin{array}{cccc} \mathbf{I}_{uv} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{I}_{TS} \end{array}\right], \tag{2.42}$$

which is a diagonal matrix with ones on the diagonal for the discrete momentum equations (2.4a-b) and the discrete tracer equations (2.4e-f) that contain a time derivative. Note that the hydrostatic pressure equation (2.4c) and the equation of conservation of mass (2.4d) do not have a time derivative, so the corresponding diagonal entries in $\mathbf{M}$ are zero.

The choice of $\Theta \in [0,1]$ defines the properties of the scheme (2.41).

If $\Theta = 0$ we obtain the explicit Forward Euler (FE) method. With FE, it is easy to compute the next states, because there are no system solves involved. However FE requires a very small time step to guarantee stability and convergence. In a typical ocean problem the maximum time step is in the order of an hour, while the timescales in the ocean can be in the order of a thousand years. Hence a model run would require an enormous amount of time steps.
If $\Theta \neq 0$ we have an implicit method. If $\Theta = 1$, we get the Backward Euler method, which is unconditionally stable, so the time step size Δt can be arbitrary large. For the effect of other choices for Θ (like $\Theta = 1/2$) see for example [78].

The solution $\mathbf{u}^{(n+1)}$ of the implicit equation is obtained by a predictor-corrector method, similar to the one we described in Section 2.3.2 for the case of numerical continuation. The Jacobian matrix involved in the corrector iterations has the form

$$\Phi_t = \frac{1}{\Delta t}\mathbf{M} + \Phi. \tag{2.43}$$

The Jacobian has the same structure as Φ in Equation (2.34). The difference between the two is that the matrices $\mathbf{A}_{uv}$ and $\mathbf{A}_{TS}$ in Φ_t are more diagonally dominant as $1/\Delta t$ is added to the diagonals. In general diagonal dominance is a pleasant property for the solvers.

For more background information on time integration see [39].

Next to straightforward implicit time stepping the data assimilation algorithm uses backward integration, which we will shortly clarify. The aim of data assimilation is to combine observations (in our case satellite data of SSH and geoid) with the numerical model (here THCM). We want to optimize the initial value for time integration such that measurements are approximated over a specified time interval as good as possible in least squares sense, leading to an error functional. For the minimization of this error functional one uses the steepest descent. Hence the gradient of the error functional has to be computed. This gradient is the integral of initial values that should be specified to an *adjoint linearized model*, such that at time t, somewhere in the specified interval, precisely the difference between measurement and the solution of our flow model based on the current initial is met. This means that first we have to solve our model on the whole interval and then from every t in the specified interval we have to integrate backwards with the adjoint linearized model to find the according initial value of that adjoint model. For details we refer to [76]. In practice it means that we have to solve systems with the transpose of the matrix Φ_t. Hence the relevance for this thesis is, that we have to take in account that the solver that we design for a Jacobian with structure (2.34) should be able to

handle the transpose of that matrix as well.

The question whether a steady solution $\mathbf{u}$ of Equations (2.4) is stable, is related to time integration. A stable solution is characterized by the fact that small distortions do not grow, but damp out as time progresses. If these distortions grow the initial solution is unstable. The growth of distortions is related to the eigenvalues and eigenvectors of the generalized eigenvalue problem

$$\Phi(\mathbf{u})\mathbf{v}_i = \lambda_i \mathbf{M}\mathbf{v}_i, \tag{2.44}$$

were the diagonal matrix $\mathbf{M}$ is given by (2.42). If all eigenvalues lie in the left half plain, i.e. $\Re(\lambda_i) < 0$, the solution is stable for all initial solutions close enough to $\mathbf{u}$. If at least one eigenvalue, say λ_1, has a positive real part the solution is unstable, i.e. starting with $\mathbf{u} + \alpha\mathbf{v}_1$, where $\alpha > 0$ may be arbitrary small, will always diverge from $\mathbf{u}$.

Qualitative changes in the eigenvalues of Φ along the branch are related to qualitative changes of the solution: bifurcation points. For example a Hopf bifurcation is equivalent to a complex eigenpair with a real part that changes of sign. At a Hopf bifurcation the solution changes from stationary to periodic. See [69, Ch.2] and [17, Ch.3] for more details.

We compute the eigenvalues and eigenvectors with the Jacobi-Davidson iteration method, presented in [72].

Chapter 3

Solving large linear systems

In this chapter we treat the techniques from numerical linear algebra needed to solve large systems of linear equations. In the previous chapter we encountered two of such systems: the Jacobian matrix in THCM in Equation (2.34) and the saddle point problem in Equation (2.36). Both systems are large and sparse, that is they have only a few nonzeros on each row. The solution of large sparse systems of equations is the domain of numerical linear algebra. In this chapter we will discuss in short the two most important approaches to solve such a system: direct methods in Section 3.1 and iterative methods in Section 3.2. In the last section of this chapter we will explain the relation of numerical continuation of solutions of THCM and the next chapters.

In the next two sections we suppose that we want to solve the equation

$$Ax = b, \tag{3.1}$$

where $A \in \mathbb{R}^{n\times n}$ and $x, r \in \mathbb{R}^n$. Furthermore we assume that n is large (typically $\mathcal{O}(10^{3-9})$) and A is sparse, i.e. per row there is a small number of nonzeros.

3.1 Direct methods

The approach in direct methods is to build a matrix factorization $A = LDU$, where L is a *unit lower triangular matrix*, which means it has ones on the diagonal and only zeros above, U a *unit upper triangular matrix* (i.e. ones on the diagonal and zeros below) and D a diagonal matrix. If A is symmetric positive definite one can build a Cholesky factorization $A = LDL^T$, so $U = L^T$.

Given such a matrix factorization, we can solve the equation in three steps: solve $Lz = b$, $Dy = z$ and $Ux = y$ respectively, which is easy, because equations with lower (or upper) triangular matrices can be solved via backward substitution.

3.1.1 Construction of an *LDU*-factorization

The *LDU*-factorization is built using Gaussian elimination. To illustrate the construction we show the elimination of the first row, where we assume that the first element of the matrix a_{11}, is nonzero:

$$A = A^{(0)} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ a_{21}a_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} a_{11} & 0 \\ 0 & A^{(1)} \end{bmatrix} \begin{bmatrix} 1 & a_{11}^{-1}a_{12} \\ 0 & I \end{bmatrix}, \tag{3.2}$$

where $A^{(1)} = A_{22} - a_{21}a_{11}^{-1}a_{12}$ is called the Schur complement of a_{11} in A. Note that $A^{(1)} \in \mathbb{R}^{n-1\times n-1}$). For $A^{(1)}$ we can build a similar factorization to obtain $A^{(2)}$. Repeating this procedure we finally get the *LDU*-factorization. It is likely that the nonzero patterns of $a_{21}a_{11}^{-1}a_{12}$ and A_{22} differ, so by the elimination we create new nonzeros in the Schur complement. In fact the growth of the number of nonzeros in the Schur complements can be huge. The factors L and U can be almost dense, even if A is sparse. Both storage (memory) and the construction time are related to the fill in the final factors, so we would like to avoid dense factors. Fortunately the fill in the factors can be reduced by simply changing the ordering of the rows and columns in A. In direct methods much effort is put in the search for a renumbering of the variables q with corresponding permutation matrix Q, such that the permuted matrix

$$QAQ^T = LDU, \tag{3.3}$$

has factors L and U that are as sparse as possible. Such an ordering q is called a *fill reducing ordering*.

3.1.2 Matrices and graphs

In the theory of fill reducing orderings graphs play an important role, see [22, 43, 52, 53], because the fill pattern of a symmetric matrix can be identified with an undirected graph. This *adjacency graph* has a number of vertices equal to the dimension of the matrix and has an edge for each off-diagonal nonzero, i.e. $\{i, j\}$ is an edge if $a_{ij} \neq 0$. In Figure 3.1 we show an example of a fill pattern of symmetric matrix ($F(A)$) and its adjacency graph ($G(A)$) for the matrix

$$A = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 4 & -1 & 0 & 0 & 0 & -1 & -1 & -1 \\ 0 & -1 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 4 & 0 & -1 & -1 \\ -1 & -1 & 0 & 0 & 0 & 0 & 4 & -1 & 0 \\ 0 & -1 & 0 & -1 & 0 & -1 & -1 & 4 & 0 \\ 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 4 \end{bmatrix}. \tag{3.4}$$

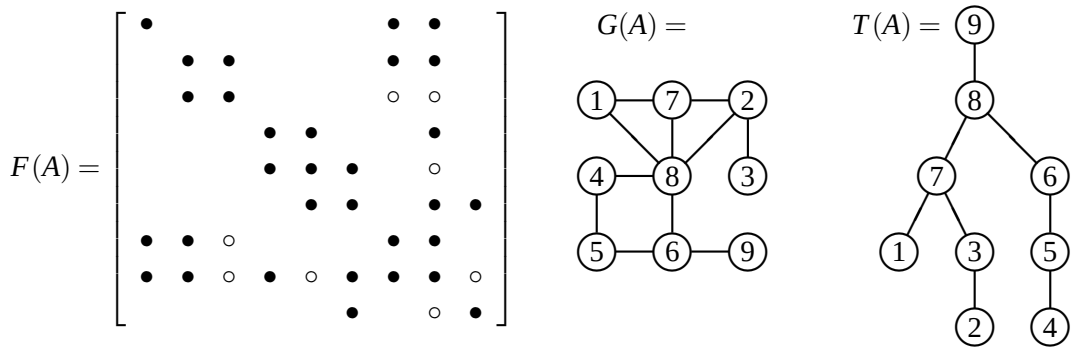


Figure 3.1: Example of the fill pattern $F(A)$ (left) of the matrix in Equation (3.4), its adjacency graph $G(A)$ (center) and the elimination tree $T(A)$ (right) for the original ordering. In $F(A)$ a $\bullet$ means a nonzero of A, a $\circ$ denotes new fill that will be created during Gaussian elimination.

A *cycle* of length l is a sequence of vertices $\{v_1, v_2, v_3, \ldots, v_l\}$ ($v_i \neq v_j$ if $i \neq j$), such that $\{v_i, v_{i+1}\}$ and $\{v_l, v_1\}$ are edges of the graph. For example the sequence $\{1,8,2,7\}$ is a cycle in $G(F)$.

A *chord* of a cycle is an edge between two non consecutive vertices, so there is an edge $\{v_i, v_j\}$, with $j \neq i \pm 1$. This chord makes it possible to split the cycle in two subcycles. The cycle $\{1,8,2,7\}$ has a chord, namely the edge $\{7,8\}$.

A *chordal graph* is a graph with the special property that every cycle with length larger than 3 has a chord. If and only if a graph is chordal, there exists a *perfect elimination ordering*, i.e. an ordering such that no extra fill is created during the factorization [52]. The search for the ordering that gives the minimum fill can be translated to the question what is the smallest number of edges that need to be added to make a graph chordal.

In the example in Figure 3.1 we have only two cycles of length larger than 3. The first is $\{1,8,2,7\}$, which has a chord. The second cycle is $\{4,5,6,8\}$ and this cycle has no chord. Consequently a matrix with the fill pattern F has no perfect elimination ordering. The creation of new fill during factorization is inevitable.

To make the graph chordal we have to add only one edge $\{4,6\}$ (or $\{5,8\}$). With this extra edge there would be a perfect elimination ordering, namely $\{1,3,2,7,9,5,4,6,8\}$. If this ordering is used for A, only two new entries will be created during factorization, while with the original ordering eight new nonzeros, the $\circ$'s in $F(A)$ in Figure 3.1, were created during Gaussian elimination. Note that each new edge connects two vertices, hence gives two nonzeros in the matrix.

The last graph shown in the figure is the *elimination tree* of the matrix. The elimination tree is a special type of graph that is constructed in the following way. Each vertex is assigned precisely one parent (see [52]):

$$\text{PARENT}[i] = \min_j \{i < j | l_{ji} \neq 0\},$$

where l_{ji} is the entry on the j-th row and i-th column of the lower triangular matrix L of Equation (3.3). So to find the parent of i we search the first nonzero lower diagonal entry in the i-th column of L. One gets the elimination tree by drawing an edge between each vertex and its parent.

Because all vertices only have one parent, we automatically get a graph without any cycles, which is called a tree. Note that one parent is allowed to have more than one child. The nodes that do not have a child are called the *leafs* of the tree. The tree $T(A)$ in Figure 3.1 has three leafs: $\{1,2,4\}$. The elimination of leafs can be done simultaneously, because they are independent from each other. This can be seen in F by the fact that the submatrix of rows and columns $\{1,2,4\}$ form a diagonal matrix.

Operations on the matrix can be translated to operations on the adjacency graph and the elimination tree. This is important for Chapter 5, so we will give an example. Let us perform the first step of Gaussian elimination on matrix (3.4). We can construct the adjacency graph of the Schur complement from the adjacency graph in Figure 3.1: we remove node $\{1\}$ and connect all its neighbours to each other. The first node has the neighbours $\{7,8\}$, which are already connected, so in this case the graph hardly changes. The graph of the Schur complement is called the *reduced adjacency graph*. The construction of the elimination tree of the Schur complement is even simpler: we cut off the leave 1. If we continue Gaussian elimination we continue pruning the tree by cutting off the leaves one by one. A nice extended paper on the role of adjacency graphs and elimination trees in sparse matrix factorization is [52].

3.1.3 Fill reducing orderings

Although the fact that the computation of the ordering that gives the minimal fill is NP-complete [89], there are good heuristic algorithms available that give good approximations for that ordering. Well-known are the nested-dissection and minimum degree ordering algorithms.

Nested-dissection [32] uses a top-down approach. It searches in a graph for the smallest set of vertices that cuts the graph in at least two components of approximately equal size. The vertices in this set should be eliminated last. The procedure is repeated for the two components. For a more detailed description of nested dissection see [53, §3.8]. In the graph $G(A)$ in Figure 3.1 nested-dissection would give node 8 the highest number in the ordering. If node 8 is removed from the graph we get two disconnected components, namely the sets of nodes $\{1,7,2,3\}$ and $\{4,5,6,9\}$.

The minimum degree approach starts at low level. The motto of this algorithm is that the vertices with the least degree (number of edges that start at a vertex) should be eliminated first [34]. In $G(A)$ in Figure 3.1 the nodes $1,3$ and 9 all have degree 1 and are assigned a low number in the new ordering. This procedure is repeated on the reduced adjacency graph. The *approximate* minimum degree algorithm [2] is a variant that uses an upper bound for the degree, which is cheaper to compute than the degree itself.

Both algorithms appear to be able to produce an ordering that gives sparse factors for huge matrices. The ordering algorithms are quite fast, because they don't use the size of the entries in A. All necessary information is contained in the adjacency graph. Based on the adjacency graph

and the ordering a symbolic factorization can be build, that tells exactly what the fill pattern of L will be. During the actual factorization only the size of the entries has to be computed, which is by far the most expensive part of the whole computation.

3.1.4 Factorizability

If A is symmetric positive definite any ordering of A will give a matrix that is factorizable, that is the factorization in (3.3) exists. However if we have an indefinite problem like the saddle point problem in Equation (2.35), the ordering has to be chosen with more care. If we blindly apply the approximate minimum degree algorithm to an indefinite matrix, it is possible that we get in trouble when the factorization is built: at some stage during Gaussian elimination as given by Equation (3.2), say at step k, we might get a Schur complement with $a_{11}^{(k)} = 0$, which was not allowed. The cause of this problem is that in the construction of the adjacency graph it was implicitly assumed that the diagonal entries in the matrix are nonzero. In Chapter 4 we will discuss the direct solution of saddle point matrices in detail. There we present a new approach to construct a fill reducing ordering for a certain class of saddle point matrices.

3.1.5 Numerical stability

Next to the existence of the factorization and the sparsity of the factors in direct methods numerical stability plays an important role. We would like to give bounds for the error in $x = L^{-T}D^{-1}L^{-1}b$, due to backward and forward substitution. As shown in [43, Ch.9] and [22, Ch.4,5], the error in x can be bounded with norms in L and D and the growth factor ρ, that is defined as

$$\rho = \frac{\max_{i,j,l} |a_{ij}^{(l)}|}{\max_{i,j} |a_{ij}|}. \tag{3.5}$$

The growth factor is the largest entry that occurs in the Schur complements during Gaussian elimination divided by the largest entry in the matrix A. In case of symmetric positive definite matrices this growth factor is 1, so there is no problem with stability. However for indefinite matrices there is no better bound than $\rho \leq 2.57^{n-1}$ (see [43, §11.1]) which is a huge number if n becomes large. In Chapter 4 we show that for a large class of saddle point problems we can compute a much smaller bound for the growth factor.

If the problem is two dimensional, direct solvers can compete in many cases with the iterative methods we will introduce in the next section. And in general direct methods are more robust than iterative methods. The disadvantage is that they often require more memory. Moreover the factorization time in direct methods is usually very high.

3.2 Iterative methods

As an alternative for the direct methods, one can use an iterative method. Instead of directly computing a solution that is as accurate as possible, the solution is approximated and at each step of the iterative method we try to improve the approximation. In this section we won't pay attention to the classical iterative methods like Jacobi, Gauss-Seidel and SOR (see [90]), because we want to focus on Krylov-subspace methods.

Given some initial guess for the solution x_0 we have an initial residual $r_0 = b - Ax_0$. The i-th Krylov space is the space spanned by

$$\mathscr{K}^i(A; r_0) = \{r_0, Ar_0, A^2 r_0, \ldots A^{i-1} r_0\}.$$

There is a wide range of Krylov subspace methods available. They have in common that the i-th approximation of x satisfies $x_i \in \mathscr{K}^i(A; r_0)$, but there are differences in the way the approximation in this space is chosen. For example GMRES chooses the x_i for which the norm $\|b - Ax_i\|_2$ is minimal over $\mathscr{K}^i(A; r_0)$, whereas the conjugate gradient method minimizes the real error in the A-norm.

For a detailed overview on Krylov subspace methods, the differences between the methods, the construction of an orthogonal basis for $\mathscr{K}^i(A; r_0)$ and theories about convergence see [82].

In this thesis we mainly use GMRES [64] or its flexible variant FGMRES [62].

3.2.1 Preconditioning

For symmetric positive definite matrices A the convergence in Krylov subspace methods is related to the eigenvalues of A, in particular the spectral condition number $\kappa = \lambda_{\max}/\lambda_{\min}$. If this number is large the convergence of x_i to x will be slow. In that case a remedy is the use of a *preconditioner* P to speed up the convergence. This preconditioner has to satisfy three properties:

(i) P is a good approximation to A in a certain sense,

(ii) P is relative cheap to construct,

(iii) the equation $Px = b$ is much easier to solve than $Ax = b$.

To clarify (i): preferably the condition number of the preconditioned matrix, i.e. $P^{-1}A$, should be small.

Again we have the problem that there is a wide range of methods that provide preconditioners: incomplete LU factorizations, multigrid methods, sparse approximate inverses, see for an overview [82, Ch.13] and [53, Ch.8].

The formulation of the preconditioner above suggests P is a matrix. However we allow P to be a more general operator on vectors. The preconditioners we describe in Chapters 6 and 7 can be iterative solvers itself. This nested iteration forces us to use a flexible Krylov method, i.e. a Krylov methods that allows variable preconditioning; for that we use FGMRES [62].

3.2.2 Incomplete LU factorization

An important class of preconditioners is formed by incomplete LU factorizations. This type of methods is based on the *LDU*-factorization as we described it in Section 3.1. The major drawback of the construction of an exact *LDU*-factorization is the growth of the fill, which has large impact on both memory requirements and construction time. The basic idea behind incomplete LU-factorizations is the reduction of the nonzeros in the Schur complements.

The general framework for the construction of an incomplete LU factorization is the following. First of all a subset of variables $x_1 \subset x$ is selected; this set may consist of a single variable. Let $x_2 = x \setminus x_1$ be the complement of this set. The matrix A is reordered according to this partitioning and split into four parts,

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}. \tag{3.6}$$

The next steps are the approximation of A_{ij} with matrices $\hat{A}_{ij}$ and the computation of the approximate Schur complement $S = \hat{A}_{22} - \hat{A}_{21}\hat{A}_{11}^{-1}\hat{A}_{12}$. In most ILU algorithms the approximation will be such, that the Schur complement is easy to compute. This is the case if $\hat{A}_{11}$ is a diagonal matrix. Now we have the following incomplete factorization for A,

$$\begin{bmatrix} I & 0 \\ \hat{A}_{21}\hat{A}_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} \hat{A}_{11} & \hat{A}_{12} \\ 0 & S \end{bmatrix}. \tag{3.7}$$

There are many variants of incomplete LU factorization. One can choose for example between a drop-by-position or a drop-by-size criterion, whether to apply Gustafsson-modification, design a sophisticated search algorithm to select the right x_1, and so on. For an overview of preconditioning methods in general and incomplete LU factorizations in particular we refer to [65] and [53, Ch.8].

Amongst the best incomplete LU factorization, there is MRILU [10]. This method is a black-box solver. It is based on matrix entries only and is comparable to algebraic multigrid.

3.3 Solving large linear systems in THCM

Except for Chapter 4 all remaining chapters in this thesis are about preconditioning. The main goal is to find a good preconditioner for the Jacobian matrix (2.34). But on the way we study preconditioners for the simpler problem (2.35).

In previous version of the thermohaline ocean model [86] equations with the Jacobian matrix (2.34) were solved with GMRES in combination with MRILU as preconditioner. MRILU used *clustering of variables*, i.e. the variables (u, v, w, p, T, S) that belong to the same cell are treated as a single variable. Although MRILU is able to build a preconditioner for the matrix, where many other methods simply fail, there is a number of disadvantages. First of all we have to choose a small drop tolerance in order to get a reasonable preconditioner. Immediate consequence is that the factorization contains a large amount of fill and therefore the construction

of the preconditioner takes much time and memory. A second drawback is that MRILU does not see the structure of the matrix, just because of the clustering. However without clustering MRILU is not able to build a good preconditioner at all, because it gets in trouble with the zero diagonal blocks.

In an overview paper on the history of iterative methods [65] the authors state: "One ray of hope for those problems that are hard to solve by iterative techniques, is to combine techniques from direct and iterative solution technologies." Hence at the start of the project we had the plan to reduce the memory requirements and construction time of MRILU by incorporation of the fill reducing orderings we discussed in Section 3.1.3. The result of the efforts to combine fill reducing orderings and incomplete LU factorizations can be found in Chapter 5. Unfortunately the results in that chapter were not as promising as we had hoped. For Poisson problems it works fine, but for coupled partial differential equations, like the Stokes equation, which gives a saddle point problem, the results are disappointing.

Chapter 4 can be considered as a byproduct of a study on direct solution methods for saddle point problems.

We searched the solution in a different direction and decided that we somehow had to exploit the structure of the matrix (2.34) in the design of a new preconditioner. As we will see in Chapter 6 for the saddle point problem and in Chapter 7 for the ocean matrix, it is useful to incorporate knowledge about different variables in the solution process. A heterogeneous system should be given a heterogeneous treatment.

Chapter 4

Numerically stable LDL^T-factorization of $\mathscr{F}$-type saddle point matrices

4.1 Introduction

In this chapter we study the direct solution of the equation

$$Kx = b, \tag{4.1}$$

where $K \in \mathbb{R}^{(n+m)\times(n+m)}$ $(n \geq m)$ is a saddle point matrix that has the form

$$K = \begin{pmatrix} A & B^T \\ B & C \end{pmatrix}, \tag{4.2}$$

with $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times n}$, and $C \in \mathbb{R}^{m\times m}$. The matrix C is symmetric and negative semidefinite, which means the eigenvalues satisfy $\lambda(C) \leq 0$. In this chapter we only consider $C = 0$ and A symmetric positive definite, so $\lambda(A) > 0$. Then the matrix K itself is symmetric, so all eigenvalues are real, and it is indefinite, i.e. there are both positive and negative eigenvalues. Although we assume A to be symmetric, many results in this chapter can be easily generalized for non-symmetric matrices A.

A survey of the occurrence of saddle point problems and their numerical solution can be found in [6]. In many cases saddle point problems can be solved efficiently via a Krylov subspace iteration [82] combined with appropriate preconditioning (see [6, 55, 27, 46] and Chapter 6 in this thesis). Nevertheless in this chapter we will focus on the direct solution of saddle point problems that occur in computational fluid dynamics. If the problem is two dimensional, direct solvers can compete in many cases with iterative methods. And in general direct methods are more robust than iterative methods. The disadvantage is that they often require more memory and that the construction of the factorization is more expensive.

The basics of sparse matrix factorization are Gaussian elimination, matrix graphs, elimination trees and fill-reducing orderings. If the reader is not familiar with these terms, we

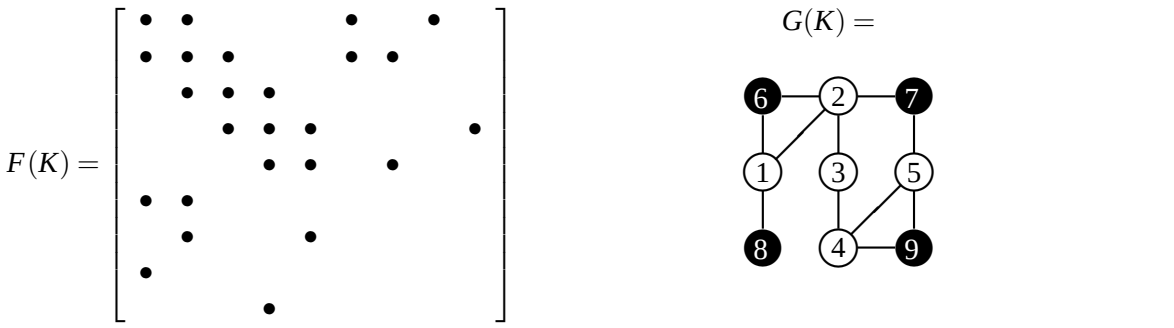


Figure 4.1: Example of the fill pattern $F(K)$ (left) and the adjacency graph $G(K)$ (right) of the matrix K as defined in Equation (4.3). In the adjacency graph the numbers of the nodes correspond to the rows of $F(K)$. The P-nodes are colored black, V-nodes are white.

strongly advise to (re)read Section 3.1 where we explained the basics of matrix factorization. In [22, 53, 43] one can find more extended introductions to this field.

Throughout this chapter we will use the following example of a saddle point problem.

Example 4.1.1. *Let K be a saddle point matrix as defined in Equation (4.1) where the matrices A, B, C are respectively*

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}, B = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \end{bmatrix}, C = 0. \tag{4.3}$$

So K is a 9×9 symmetric saddle point matrix.

The nodes of the adjacency graph of a saddle point matrix K can be divided in two sets: V and P. The set V contains all the nodes that correspond to the first n rows of K. V contains precisely the nodes that stem from the submatrix A. In fluid problems these are the velocity nodes. The set P contains all the nodes that originate from the last m rows in K (the block $C = 0$). In fluid terms these are the pressure nodes. The nodes in V and P will be called V-nodes and P-nodes respectively.

In Figure 4.1 one finds the fill pattern and the adjacency graph of the matrix in Example 4.1.1. For this example $V = \{1,2,3,4,5\}$ and $P = \{6,7,8,9\}$. The P-nodes are colored black in the adjacency graph.

The aim of this chapter is to find a fill-reducing pre-ordering Q such that we can perform Gaussian elimination on the permuted saddle point matrix QKQ^T, resulting in a factorization of the form

$$QKQ^T = LDL^T, \tag{4.4}$$

where L is a unit lower triangular matrix and D a block-diagonal matrix with blocks of size 1×1 or 2×2.

The search for such an ordering Q is far from trivial for large sparse matrices in general and especially not in the case of symmetric indefinite saddle point matrices. There are three major issues that we have to address about the factorization (4.4):

(1) *factorizability*: given an ordering Q, does the factorization exist?
(2) *sparsity*: can we construct Q such that it reduces the fill in the L-factor?
(3) *numerical stability*: given an ordering Q and the corresponding LDL^T-factorization, can we prove that the errors in $x = L^{-T}D^{-1}L^{-1}b$ are bounded?

We start with (1), the factorizability. Positive definite matrices like our submatrix A are *strongly factorizable*, i.e. all possible orderings give a factorizable matrix. This does not hold for saddle point matrices. In Example 4.1.1 we can choose the Q such that it is the reordering matrix corresponding to the permutation $q = \{6,7,8,9,1,2,3,4,5\}$. This would give the following permuted saddle point matrix

$$\begin{bmatrix} 0 & B \\ B^T & A \end{bmatrix},$$

which is not factorizable, because if we start performing Gaussian elimination as in Equation (3.2), we get in the first step $a_{11} = 0$. The minimum degree ordering for the matrix (computed with matlab) is $q_m = \{6,8,1,3,5,9,4,2,7\}$. This ordering has the same problem as it wants to eliminate the black nodes 6 and 8 first. In fact any ordering that starts with a P-node will suffer from this problem. In general a node in P cannot be eliminated before one of its neighbours (which are all in V, because $C = 0$) is eliminated. Or, equivalently, as formulated in [80], a necessary condition for factorizability is that the elimination tree of the permutated matrix has no leaves in P. Let Q_m be the permutation matrix corresponding to q_m, then the elimination tree for the matrix $Q_m K Q_m^T$ has three leaves in P as is shown in Figure 4.2 (left), so the reordered saddle point matrix is not factorizable.

The example shows that the ordering for saddle point matrices has to be chosen carefully. An ordering that gives a factorizable permuted matrix is called *feasible*.

Amongst saddle point problems there are only few exceptions. In [83] the author considers the factorization of quasi-definite symmetric saddle point matrices: A is symmetric positive definite ($\lambda(A) > 0$) and C symmetric negative definite ($\lambda(C) < 0$). Quasi-definite saddle point matrices are strongly factorizable, so any ordering is feasible.

Most literature is about the factorization of sparse symmetric indefinite matrices in general [25, 23], which is a much larger class than the saddle point problems that we consider. To produce a feasible ordering, algorithms for factorization of indefinite matrices often use the pivoting strategies of Bunch-Parlett [13] or Bunch-Kaufman [12]. These strategies are applied in the factorization phase and basically do the following: if a zero pivot like $a_{11} = 0$ is encountered, a search is started for a second node that is coupled to the first, such that the two together

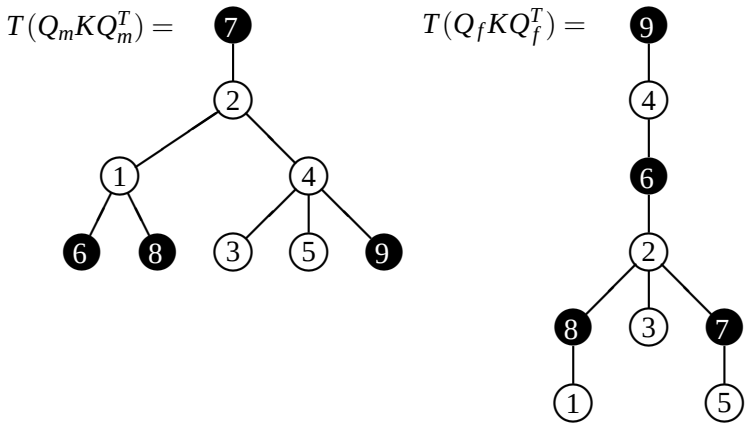


Figure 4.2: Two elimination trees for the saddle point matrix in Example 4.1.1. The tree at the left belongs to the minimum degree ordering, the tree at the right to the ordering given by Algorithm 4.2.2.

form a stable and invertible 2×2 matrix. In case of our example, the nodes $\{6,2\}$ would form an acceptable pivot, because

$$\begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix}$$

is invertible. Because of the Bunch-Parlett and Bunch-Kaufman pivoting strategies, the matrix D in (4.4) is allowed to have 2×2 blocks on the diagonal.

Recently these pivotting strategies were fruitfully combined with a weighted matchings search algorithm in the package PARDISO [61, 38, 67].

The only paper that really focusses on the factorization of our type of saddle point matrices (with $C = 0$) is [80]. We will return to the contents of that paper at the end of this section.

The second important issue is (2) the sparsity of the factors. To reduce both memory requirements and construction time we would like the factor L to be as sparse as possible. In Section 3.1 we introduced the two most important algorithms to compute a fill-reducing ordering for a matrix: approximate minimum degree [2] and nested dissection [32]. Unfortunately they only apply to positive definite matrices, because they are based on the adjacency graph of the matrix, which assumes a positive diagonal entry. The algorithms do not see the difference between white (V) and black (P) nodes in Figure 4.1. In our example the approximate minimum degree ordering gives a non feasible ordering, because the black node 8 (that has the smallest number of neighbours) is selected first in the factorization phase.

If the approximate minimum degree algorithm is applied to a general saddle point matrix K it is unlikely that the computed ordering is feasible. So the ordering has to be repaired either during elimination, by delaying the elimination of P-nodes, or by adapting the ordering beforehand. Repair during elimination is expensive because we have to check every pivot. In [80] the ordering is repaired before elimination. After the computation of a fill-reducing ordering for K the ordering is adapted based on elimination tree operations solely. For so called $\mathscr{F}$-matrices,

to be introduced later in this section, it can be proven that the final ordering is feasible. However delay of elimination of indefinite nodes either during elimination or beforehand creates extra fill in the factor L.
There is one more alternative, that is called the Schur complement approach. The elimination of the nodes from P is delayed until all V-nodes are eliminated. Unfortunately in many cases the Schur complement $C - BA^{-1}B^T$ is completely full. So this approach is not very practical and we won't spend attention to this approach.

The last issue is (3) numerical stability. In Section 3.1 we introduced the growth factor

$$\rho = \frac{\max_{i,j,l} |k_{ij}^{(l)}|}{\max_{i,j} |k_{ij}|}, \tag{4.5}$$

which is an important measure for stability. The growth factor is the largest entry that occurs in the Schur complements during Gaussian elimination divided by the largest entry in the matrix K. This growth factor can become very large, even if we have a feasible ordering for K. If we consider Bunch-Kaufmann or Bunch-Parlett pivoting the stability bound for the growth factor is very weak: $\rho \leq 2.57^{(n+m-1)}$ [43, §11.1]. Although in many applications no problems were reported with numerical stability there is lack of better bounds for ρ. One of the important results in this chapter is that we were able to give a much better bound for the growth factor in case of $\mathscr{F}$-matrices.

These $\mathscr{F}$-matrices are a special class of saddle point matrices that play an important role in the rest of the chapter as well. We give the definition in two steps.

Definition 4.1.2. *A gradient-type matrix has at most two entries per row. Moreover, if there are two entries, their sum is zero.*

We have chosen the name gradient-type matrix, because this type of matrix typically results from the discretization of a pressure-gradient in flow equations. It is important to note that the definition allows a gradient-type matrix to be non-square. We use gradient-type matrices to define the $\mathscr{F}$-matrices.

Definition 4.1.3. *An $\mathscr{F}$-matrix is a saddle point matrix (4.2) with $C = 0$, A symmetric and positive definite and B^T a gradient-type matrix.*

The original definition is of Tůma in [80]. $\mathscr{F}$-matrices occur in various fluid flow problems where Arakawa A-grids (collocated) or C-grids (staggered, see Figure2.2) are used. They occur as well in electrical networks [84]. Note that the saddle point matrix in Example 4.1.1 is an $\mathscr{F}$-matrix.

Given a feasible, numerically stable ordering for K there are several efficient codes available for construction of the corresponding LDL^T factorization. We mention MA47 [23] as an

example, but we won't pay much attention to the actual factorization of the permutated matrix.

The outline of the chapter is as follows. In the next section we sketch the algorithm to compute a fill-reducing ordering for a saddle point matrix. In Section 4.3 we show properties that remain invariant under Gaussian elimination with this ordering. In Section 4.4 we give a proof for numerical stability of Gaussian elimination for $\mathscr{F}$-matrices using this ordering. Symbolic factoring of $\mathscr{F}$-matrices is treated in Section 4.5. In Section 4.6 we show the numerical results for a Stokes equation in a driven cavity and for a set of $\mathscr{F}$-matrices that is used in [80]. And we end with a discussion in Section 4.7.

4.2 Sketch of the algorithm

The algorithms that we mentioned in the previous section have in common that they compute a fill-reducing ordering for K and then somehow adapt it to make it feasible. Especially if we deal with saddle point problems where B has fewer entries per row than A - which is often the case for $\mathscr{F}$-matrices - a fill-reducing ordering like (approximate) minimum degree will choose the nodes in P to be eliminated first, because they have the lowest degree. Tůma [80] wrote that for his matrices 80% of the leaves of the elimination tree belongs to the set P. For all these leaves the elimination has to be delayed.

To overcome this inefficiency we propose a different approach. The idea is to compute an ordering for V only and then insert the nodes in P under the following condition.

Condition 4.2.1. *If during Gaussian elimination with K the node $v \in V$ is to be eliminated and it is connected to a $p \in P$ then v and p are eliminated together using a 2×2 pivot.*

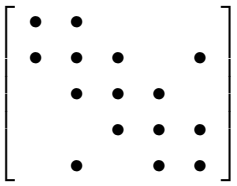
This simple rule is used to insert all the nodes of P in the ordering of V which gives us an ordering for K. Note that in this way we get as much 2×2 pivots as there are nodes in P. Only if a node $v \in V$ becomes totally disconnected from P due to elimination of previous nodes it can be eliminated solely.

Because all P-nodes are eliminated together with a V-node in pivots of the form

$$\begin{pmatrix} a & b \\ b & 0 \end{pmatrix},$$

there is no doubt about factorizability. We immediately get a feasible ordering, so we don't need any (expensive) additional repairs.

If we apply this procedure to a fill-reducing ordering for A the resulting ordering will not be fill-reducing for K in general. However if we apply the procedure to a fill-reducing ordering of $A + B^T B$, it will generate a fill-reducing ordering for K, because the fill pattern $A + B^T B$ is an envelope for the fill that is created by eliminating the nodes in P. If the zero block was not empty, but a diagonal matrix (for example $-I$) we would get the fill pattern of this matrix after elimination of the nodes in P. To prevent exact cancellation we will use the matrix $F(A) +$

$F' =$ 

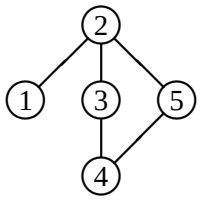
$G(F') =$ 

Figure 4.3: The fill pattern $F' = F(A) + F(B^T)F(B)$ (left) and the corresponding adjacency graph (right) that is used to compute an ordering for the V-nodes of Example 4.1.1.

$F(B^T)F(B)$, where $F(A)$ denotes the fill pattern of A. Summarizing we get the following algorithm.

Algorithm 4.2.2. *To compute a feasible fill-reducing ordering for saddle point matrix K:*

1. *Compute a fill-reducing ordering for the V-nodes based on $F(A)+F(B^T)F(B)$.*
2. *Insert the P-nodes into the ordering subject to Condition 4.2.1.*

The P-nodes (step 2) can be inserted during Gaussian Elimination, which means that we have to adapt the algorithm of Gaussian elimination. However in case of $\mathscr{F}$-matrices it can be done symbolically before elimination. The very efficient algorithm for that is described in Section 4.5.

Let us apply the algorithm to the matrix in Example 4.1.1. In the first step we will compute an ordering for the V-nodes that is based on the fill pattern $F(A)+F(B^T)F(B)$.

This fill pattern and the adjacency graph can be found in Figure 4.3. The minimum degree ordering for this matrix is $q_v = \{1,3,5,2,4\}$. The second step is the insertion of the P-nodes in this ordering. The first node in the ordering ($q_v(1) = 1$) is connected to the P-nodes 6 and 8 (see Figure 4.1), so according to Condition 4.2.1 we have to choose one of these two nodes to eliminate together with node 1. For now this choice is arbitrary, but in Section 4.5 we will argue that we better choose node 8, because it has the least degree (number of neighbours) of the two. If we remove nodes 1 and 8 from the graph and continue the elimination and insertion of P-nodes, we finally get the ordering $q_f = \{1,8,3,5,7,2,6,4,9\}$. With this ordering the permuted saddle point matrix becomes (we leave out the zeros)

$$Q_f K Q_f^T = \begin{bmatrix}
2 & 1 & & & & -1 & -1 & & \\
1 & & & & & & & & \\
 & & 2 & & & -1 & & -1 & \\
 & & & 2 & 1 & & & -1 & \\
 & & & 1 & & -1 & & & \\
-1 & & -1 & & -1 & 2 & 1 & & \\
-1 & & & & & 1 & & & \\
 & & -1 & -1 & & & & 2 & -1 \\
 & & & & & & & -1 &
\end{bmatrix}, \tag{4.6}$$

and its factors are

$$L = \begin{bmatrix} 1 & & & & & & & & \\ \frac{1}{2} & 1 & & & & & & & \\ & & 1 & & & & & & \\ & & & 1 & & & & & \\ & & & \frac{1}{2} & 1 & & & & \\ -\frac{1}{2} & -1 & -\frac{1}{2} & & 2 & 1 & & & \\ -\frac{1}{2} & -1 & & & & \frac{2}{7} & 1 & & \\ & & -\frac{1}{2} & -\frac{1}{2} & -1 & -\frac{3}{7} & -\frac{3}{2} & 1 & \\ & & & & & & & 1 & 1 \end{bmatrix},$$

$$D = \begin{bmatrix} 2 & & & & & & & & \\ & -\frac{1}{2} & & & & & & & \\ & & 2 & & & & & & \\ & & & 2 & & & & & \\ & & & & -\frac{1}{2} & & & & \\ & & & & & \frac{7}{2} & & & \\ & & & & & & -\frac{2}{7} & & \\ & & & & & & & \frac{3}{2} & \\ & & & & & & & & -\frac{2}{3} \end{bmatrix}.$$

We chose to build the factorization such that D is a diagonal matrix with no 2×2 blocks on the diagonal. Consequently D has negative entries on the diagonal. This factorization allows us to construct the elimination tree that is based on the L-factor (for definition see Section 3.1). The tree is depicted in Figure 4.2 (right).

The motivation for this approach can be understood from a nested-dissection point of view. In our type of applications often B is a discrete divergence and B^T a discrete gradient operator. The rows of B then represent the continuity equation in each cell of the grid. If we deal with a C-type staggered grid (e.a. the pressure nodes are in cell centers, the velocity nodes on cell faces, see Figure 2.2) it means that all velocity nodes occur in the continuity equation of two different cells. To compute the pressure gradient in a velocity node we take the difference of the pressure values in the two neighbouring cells. The elimination of a velocity node forces the two neighbouring cells to merge. We need only one pressure node to fix the pressure in a cell, so we can eliminate one of the two together with the velocity node. Or equivalently we can eliminate one of the two continuity equations, because the pressure nodes in the two cells get connected by the elimination of the velocity node. In this view we can consider the velocity nodes as separators of the pressure nodes, that need to be eliminated as soon as possible. That is precisely what is formulated in Condition 4.2.1 and is used in Algorithm 4.2.2.

In the rest of the chapter $K^{(l)}$ denotes the matrix K after l steps of Gaussian elimination according to an ordering defined by the algorithm above. If it is not necessary we won't make a distinction between $K^{(l)}$ or any symmetric permutation of the matrix.

4.3 Properties invariant under Gaussian elimination

In this section we will show that Algorithm 4.2.2 gives an ordering with the nice property that during elimination much of the structure of the original matrix is preserved. If we perform Gaussian elimination on the matrix K we get a sequence of Schur-complements $K^{(l)}$ for $l = 1,\ldots,n$. If we separate the V- and P-nodes, the structure of all these matrices is

$$K^{(l)} = \begin{pmatrix} A^{(l)} & B^{(l)T} \\ B^{(l)} & C^{(l)} \end{pmatrix}.$$

The first important property is that Gaussian elimination subject to Condition 4.2.1 on a saddle point matrix (4.2) with $C = 0$ always gives a saddle point matrix with the very same structure as Schur complement. In other words: the nodes in P are not coupled in K and will never get coupled in $K^{(l)}$.

Theorem 4.3.1. *If $C^{(0)} =$ then for all l we have $C^{(l)} = 0$.*

Proof. The proof is by induction. Suppose $C^{(l)} = 0$. Let us compute $K^{(l+1)}$ by eliminating the first row in $K^{(l)}$. Now we have to distinguish two cases that can occur.

The first possibility is that the l-th node *is not* connected to a node in P. We can make this more explicit by splitting $A^{(l)}$ and $B^{(l)}$:

$$K^{(l+1)} = \begin{pmatrix} a & \alpha^T & 0 \\ \alpha & \hat{A} & \hat{B}^T \\ 0 & \hat{B} & 0 \end{pmatrix}.$$

According to condition 4.2.1 this node can be eliminated solely. The Schur complement of a is equal to $K^{(l+1)}$ and easy to compute

$$K^{(l)} = \begin{pmatrix} A^{(l+1)} & \hat{B}^T \\ \hat{B} & 0 \end{pmatrix}. \tag{4.7}$$

with

$$A^{(l+1)} = \hat{A} - \alpha^T \alpha / a. \tag{4.8}$$

Clearly $C^{(l+1)} = 0$.

The other possibility is that the l-th node *is* connected to a node in P. Then the structure is:

$$K^{(l)} = \left(\begin{array}{cc|cc} a & b & \alpha^T & \beta^T \\ b & 0 & \hat{\beta}^T & 0 \\ \hline \alpha & \hat{\beta} & \hat{A} & \hat{B}^T \\ \beta & 0 & \hat{B} & 0 \end{array}\right).$$

To satisfy condition 4.2.1 the first two rows are eliminated together using a 2×2 pivot. Elimination gives

$$K^{(l+1)} = \begin{pmatrix} A^{(l+1)} & \hat{B}^T - \hat{\beta}(\beta/b)^T \\ \hat{B} - (\beta/b)\hat{\beta}^T & 0 \end{pmatrix}, \tag{4.9}$$

where

$$A^{(l+1)} = \hat{A} - \alpha(\hat{\beta}/b)^T - (\hat{\beta}/b)\alpha^T + a(\hat{\beta}/b)(\hat{\beta}/b)^T. \tag{4.10}$$

Also in this case we have $C^{(l+1)} = 0$. □

In the proof of the theorem we see that in both cases $B^{(l+1)}$ is determined by the elements of $B^{(l)}$ only. No elements of A are involved.

Corollary 4.3.2. *For all l the matrix $B^{(l)}$ is independent of the fill pattern and the size of the entries in A. It only depends on the ordering for the V-nodes.*

Immediate consequence of this Corollary is that we can compute the sequence of $B^{(l)}$'s without using the size of the entries of A. We can exploit this in the symbolic factoring phase to insert the nodes of P into the ordering of V. This will return in Section 4.5.

The following theorem is important for stability.

Theorem 4.3.3. *If A is symmetric positive definite, then all $A^{(l)}$ are symmetric positive definite.*

Proof. This is easy to prove by induction. We have $A^{(0)} = A$ symmetric positive definite by assumption. Now assume $A^{(l)}$ is symmetric positive definite. Elimination of the next node will gives us $K^{(l+1)}$ and $A^{(l+1)}$. As in the proof of Theorem 4.3.1 we have to distinguish two cases.

If the l-th node *is not* coupled to a node in P, $A^{(l+1)}$ is given by Equation (4.8). It is the Schur complement of a symmetric positive definite matrix which is again symmetric positive definite.

If the l-th node *is* coupled to a node in P, $A^{(l+1)}$ is given by Equation (4.10). We can rewrite it to

$$A^{(l+1)} = \hat{A} - \alpha^T\alpha/a + a(\alpha/a - \hat{\beta}/b)(\alpha/a - \hat{\beta}/b)^T. \tag{4.11}$$

It is the Schur-complement of a positive definite matrix $(\hat{A} - \alpha^T\alpha/a)$ plus the term $a(\alpha/a - \hat{\beta}/b)(\alpha/a - \hat{\beta}/b)^T$ which is obviously symmetric and positive definite if $a > 0$. Because $A^{(l)}$ is symmetric positive definite all its diagonal elements are positive, so we have indeed $a > 0$. □

Theorem 4.3.4. *If K is an $\mathscr{F}$-matrix, all $K^{(l)}$ are $\mathscr{F}$-matrices.*

Proof. By assumption we know $K^{(0)} = K$ is an $\mathscr{F}$-matrix. According to Definition 4.1.3 it holds that $C^{(0)} = 0$, $A^{(0)}$ is symmetric positive definite and $B^{(0),T}$ is a gradient-type matrix. The Theorems 4.3.1 and 4.3.3 respectively ensure $C^{(l)} = 0$ and $A^{(l)}$ symmetric positive definite, so we only have to prove that $B^{(l),T}$ is a gradient-type matrix.

Once more we use induction. So let us assume that $B^{(l),T}$ is a gradient-type matrix. Again we distinguish the two cases that occur during Gaussian elimination.

If the l-th node *is not* coupled to a pressure node we have according to Equation (4.7)

$$B^{(l+1),T} = \hat{B}^T.$$

The only difference with $B^{(l)T}$ is the omission of the first (empty) row. So properties like the number of entries per row and row sum are certainly preserved and $B^{(l+1),T}$ is a gradient-type matrix.

If the l-th node *is* coupled to a pressure node we have according to Equation (4.9)

$$B^{(l+1),T} = \hat{B}^T - \hat{\beta}(\beta/b)^T. \tag{4.12}$$

The gradient-type matrix $B^{(l)}$ has at most two entries per row; when a row has precisely two entries, they have zero sum. The first row of $B^{(l)}$ is $(b \;\; \beta^T)$. Because b is nonzero, β^T has either one entry with value $-b$ or no entries at all. Hence $-\beta^T/b$ is either a vector with one entry with value 1 or a zero vector. In Equation (4.12) $\hat{\beta}$ (i.e. the first column of $B^{(l),T}$) is multiplied with this vector and added to $\hat{B}^T$. This leads to the following simple procedure to construct $B^{(l+1),T}$. We get $B^{(l+1),T}$ from $B^{(l),T}$ by removing its first row and first column ($\hat{\beta}$) - which gives $\hat{B}^T$ - and add $\hat{\beta}$ to the column that had an entry on the first row (β^T). Obviously the rows with no entry in the first column do not change at all, so the number of entries and the row sum are preserved. Of interest are the rows that have an entry in $\hat{\beta}$. Now there are two possibilities. If $\beta^T = 0$ the only change is that the number of entries on the row decreases with one and the row sum does not matter anymore, because there is either one or no entry left. In that case $B^{(l+1),T}$ is a gradient-type matrix. Otherwise (β^T has precisely one entry) the row sum is preserved, because we multiply the first column with 1 and add it to one of the other columns of the matrix $\hat{B}^T$. So now $B^{(l+1),T}$ is a gradient-type matrix as well. □

Remark 4.3.5. Exact cancellation can occur in $B^{(l+1),T}$ during Gaussian elimination. Fortunately we precisely know when this happens, namely if there is a row (not the first one) in $B^{(l),T}$ that is a multiple of the first row. As shown in the last proof the entries on that row get summed up in $B^{(l+1),T}$ and cancel each other out, because their sum is zero. It is advantageous to know when cancellation happens, because this allows us to insert the nodes beforehand in the ordering in the symbolic factoring phase as we will see in Section 4.5.

Remark 4.3.6. For $B^{(l),T}$ the size of the entries does not change. Each row in $B^{(l),T}$ can be traced back to a row in $B^T = B^{(0),T}$. If the row in $B^{(l),T}$ is not empty, the size of the entries (at most two) on both rows is exactly the same. This is an immediate consequence of the proof of the last theorem. Entries on a row can move and possibly coincide in which case they annihilate, but the size never changes. So we have $\max_{ij} |b_{ij}^{(l)}| \leq \max_{ij} |b_{ij}^{(0)}|$. This is very useful in the next Section, where we will consider numerical stability of Gaussian elimination. The B's do not contribute to the growth factor.

Summarizing we can say that with Condition 4.2.1 a lot of structure of the original saddle point matrix is preserved during Gaussian elimination.

4.4 Numerical stability

In the introduction we already mentioned the growth factor ρ (see (4.5)). A bounded growth factor is important for numerical stability in Gaussian elimination and for the backward error in the application of the factorization. See [43, Ch.9] and [22, Ch.4,5] for more details.

The pivoting strategies of Bunch-Parlett and Bunch-Kaufmann that apply for the indefinite case (they use 2×2 pivots as well) guarantee the bound $\rho \leq (2.57)^{(n+m-1)}$ [43, §11.1]. This bound is very weak. It grows exponentially with the problem size. Fortunately in many cases the growth factor stays far away from the upper bound. But in general it is hard to find a substantially smaller upper bound. At least we do not know any literature that computes a better bound in case of saddle point matrices.

We found such a smaller bound for $\mathscr{F}$-matrices. In this section we present a theorem and proof for a bound that depends only linearly on m.

There are two reasons why we are able to compute a better bound. First of all we do not study symmetric indefinite matrices in general: we restrict ourselves to the class of $\mathscr{F}$-matrices. We explicitly use the special properties of $\mathscr{F}$-matrices (see Definition 4.1.3) in the proof. The second reason is the way we compute the bound. The bound $(2.57)^{(n+m-1)}$ is computed looking at the elimination of a single node. So a bound for the elements in $K^{(l+1)}$ is based on the size of the elements in $K^{(l)}$ only. With the assumption of complete pivoting (Bunch-Parlett) or partial pivotting (Bunch-Kaufman) one gets a bound for the growth of the elements in a single step of Gaussian elimination. The bound for this "local" growth factor is $\sqrt{1+2/(1-\alpha)}$, where $\alpha = (1+\sqrt{17})/8$, which gives the number 2.57. Realizing that we have to compute a total of $(n+m-1)$ Schur complements, we get the bound for the overall growth factor: $\rho \leq (2.57)^{(n+m-1)}$. Any method that computes a local growth factor that is bigger than 1 gives an exponentially growing growth factor. The only way to avoid this is a global approach: we compute a bound for the elements of $K^{(l+1)}$ directly in terms of the elements of $K^{(0)}$. Instead of focussing on the effect of the elimination of the last node only we will look at the effect of the elimination of all previous nodes. To allow this we need some of the properties of $\mathscr{F}$-matrices and the fact that saddle point matrices preserve much of their structure under Gaussian elimination with Condition 4.2.1, as we showed in the previous section.

Before we give the theorem we remark that in this section we assume that all entries in B have a value in $\{-1,0,1\}$. This is not a restriction because if B doesn't satisfy this property it can be forced to do so by a simple column scaling.

Theorem 4.4.1. *Gaussian elimination on an $\mathscr{F}$-matrix with a feasible ordering is numerically stable. Let σ be the maximum of the number of entries per row in A. Furthermore let m be the number of rows in B then the stability bound is*

$$\rho \leq (2+m)\sigma - 1.$$

This bound is much better than the general one for indefinite matrices. It grows only linearly with m instead of exponentially. We give the proof of the theorem at the end of the section. To simplify the proof we need a few lemmas on gradient-type matrices.

Lemma 4.4.2. *Let U be a unit upper triangular gradient-type matrix then U^{-1} is unit upper triangular and all its entries have a value* 1 *or* 0.

Proof. Let V be the inverse of U. Furthermore let v be the k-th column of V and e the k-th unit vector. It holds that $v = Ve = U^{-1}e$. Because U is upper triangular we can use backward substitution to compute v. The general rule for backward substitution is: $v_i = (e_i - \sum_{j=i+1}^{n} u_{ij}v_j)/u_{jj}$ for $i = n$ down to 1. Here the rule simplifies a lot because U is a gradient-type matrix and e is a unit vector. Because U is an unit upper triangular gradient-type matrix we have for the diagonal entries $u_{jj} = 1$ and there is at most one nonzero u_{ij} and its value has to be -1. Furthermore, because e is the k-th unit vector, $e_i = 0$ if $i \neq k$ and $e_k = 1$. The rules for backward substitution in this special case are

$$
\begin{aligned}
&(1)\ v_i = \quad 0 \quad \text{for } i = n \text{ down to } k+1,\\
&(2)\ v_i = \quad 1 \quad \text{if } i = k,\\
&(3a)\ v_i = \quad 0 \quad \text{for } i = k-1 \text{ down to } 1, \text{ if there is no upper diag. entry on row } U_{i*},\\
&(3b)\ v_i = \quad v_j \quad \text{for } i = k-1 \text{ down to } 1, \text{ if } u_{ij} = -1 \text{ is the upper diag. entry on row } U_{i*},.
\end{aligned}
$$

Note that we do backward substitution, so we start with $i = n$ and then go down to $i = 1$. Consequently at first rule (1) is applied a couple of times, then rule (2) is applied once, and thereafter only the rules (3a) and (3b) are applied. So we immediately have $v_i = 0$ if $i > k$ and $v_k = 1$. For all v_i with $i < k$ have a value that is assigned by either (3a) or (3b). If rule (3b) is used we have $v_i = v_j$, where $j > i$, so v_j got a value before v_i. We can find this value looking at the upper diagonal entry at row U_{j*}. If it has a nonzero entry $u_{jj_2} = -1$ then $j_2 > j$ and $v_j = v_{j_2}$. We can repeat this until we get an index j_N that is assigned a value by one of the rules (1),(2) or (3a), so $v_{j_N} = 0$ or 1. Because $v_i = v_j = v_{j_2} = \cdots = v_{j_N}$, the same holds for v_i. It follows that v contains ones and zeros only. But v is an arbitrary column of $V = U^{-1}$, so the inverse of U contains ones and zeros only. □

To illustrate the proof, we give below a matrix U and its inverse

$$
U = \begin{pmatrix} 1 & 0 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad U^{-1} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.
$$

Lemma 4.4.3. *Let G be an invertible gradient-type matrix with values in $\{-1,0,1\}$. There exist permutation matrices Q_1 and Q_2 and a diagonal matrix D with entries ± 1 such that $U = DQ_1GQ_2$ is a unit upper triangular gradient-type matrix.*

Proof. The most important concern is to get it in upper triangular form. Because G is an invertible gradient-type matrix it contains at least one row with only one entry. Let's assume that it is the i-th row that has a single entry in the j-th column. We can permute the matrix such that the i-th row becomes the last row and the j-th column becomes the last column. Let

u be the j-th column of G where we removed the entry $d = g_{ij}$ and let G' be the matrix that is obtained from G by removing its i-th row and the j-th column. Now the matrix

$$\begin{pmatrix} G' & u \\ 0 & d \end{pmatrix}$$

is a permutation of G. Note that G' itself is an invertible gradient-type matrix so we can apply the very same procedure to G' which gives a submatrix G'' which again is an invertible gradient-type matrix. Repeated application of the procedure will end with a $G^{(n-1)}$ of dimension 1. This gives us an upper triangular matrix $\tilde{U} = Q_1 G Q_2$ with entries ± 1 on the diagonal. Choose D equal to the diagonal of $\tilde{U}$ and $U = D\tilde{U} = DQ_1 G Q_2$ is a unit upper triangular gradient-type matrix. □

Lemma 4.4.4. *Let G be an invertible gradient-type matrix with entries in $\{-1,0,1\}$. Then all entries of G^{-1} are in $\{-1,0,1\}$ as well.*

Proof. We simply combine the previous two lemmas. Lemma 4.4.3 states that $G = Q_1^T D U Q_2^T$ with D a diagonal matrix with ± 1 on the diagonal, Q_1 and Q_2 permutation matrices and U a unit upper triangular gradient-type matrix. According to Lemma 4.4.2 U^{-1} consists of ones and zeros, so $G^{-1} = Q_2 U^{-1} D Q_1$, which is nothing but a permutation and scaling with ± 1 of U^{-1}, consists of the values $\{-1,0,1\}$. □

To simplify the notation in the rest of this section we will use $\mu(A)$ to denote the largest entry in the matrix A, the formal definition being

$$\mu(A) = \max_{ij} |a_{ij}|.$$

Note that obviously we have $\mu(A^T) = \mu(A)$. Using this notation the growth factor becomes

$$\rho = \max_l \mu(K^{(l)}) / \mu(K). \tag{4.13}$$

Furthermore we introduce $\sigma(A)$ to denote the maximum number of nonzero entries per row in A. So each row in A has at most $\sigma(A)$ nonzeros. Consequently the number $\sigma(A^T)$ denotes the maximum number of nonzeros per column in A.

Note that both $\mu(A)$ and $\sigma(A)$ give some information about the nonzero entries of A: $\mu(A)$ about their *magnitude* and $\sigma(A)$ about their *position.*

In the following we derive some properties for μ.

Lemma 4.4.5. *Let $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{m\times l}$ then*

$$\begin{aligned} \mu(AB) &\leq \sigma(A)\mu(A)\mu(B), \\ \mu(AB) &\leq \sigma(B^T)\mu(A)\mu(B), \\ \mu(AB) &\leq m\mu(A)\mu(B). \end{aligned}$$

Proof. The second bound follows from the first if we note that $\mu(AB) = \mu((AB)^T) = \mu(B^T A^T)$. Because $\sigma(A) \leq m$ and $\sigma(B^T) \leq m$ the third bound is easily obtained as well, so we only have to prove the first bound. This follows from

$$\mu(AB) = \max_{ij} \left| \sum_{k=1}^{m} a_{ik} b_{kj} \right| \leq \max_{ij} \sum_{k=1}^{m} |a_{ik}||b_{kj}| \leq \mu(B) \max_{i} \sum_{k=1}^{m} |a_{ik}| \leq \mu(B)\sigma(A)\mu(A). \quad (4.14)$$

□

If we deal with a diagonally dominant matrix we have a sharper bound.

Lemma 4.4.6. *Let $A \in \mathbb{R}^{n\times m}$ be diagonally dominant and $B \in \mathbb{R}^{m\times l}$ then*

$$\mu(AB) \leq 2\mu(A)\mu(B).$$

Proof. The matrix A is diagonally dominant, so $\sum_{k=1}^{m} |a_{ik}| \leq 2|a_{ii}| \leq 2\mu(A)$. If we use this in the last step in (4.14) in the proof of Lemma 4.4.5 we get the desired expression. □

If at least one of the two matrices in a matrix product is a gradient-type matrix we can say more about the size of the elements of the product.

Lemma 4.4.7. *Let $G \in \mathbb{R}^{n\times m}$ be a gradient-type matrix with entries in $\{-1, 0, 1\}$ and let $A \in \mathbb{R}^{m\times l}$. If $A \leq 0$ or $A \geq 0$ then*

$$\mu(GA) \leq \mu(A).$$

Proof. This inequality follows immediately from the properties of a gradient-type matrix as given in Definition 4.1.2. An entry of GA is either equal to an entry of A or equal to the difference of two entries of A of equal sign, so $\mu(GA) \leq \mu(A)$. □

Lemma 4.4.8. *Let $G_1 \in \mathbb{R}^{n\times m}$ and $G_2 \in \mathbb{R}^{m\times m}$ be gradient-type matrices with entries in ± 1. Let G_2 be invertible then*

$$\mu(G_1 G_2^{-1}) \leq 1.$$

Proof. The matrix G_2 is an invertible gradient-type matrix with entries in ± 1, so we can apply Lemma 4.4.4 and obtain a factorization $G_2^{-1} = Q_2 U^{-1} D Q_1$ where Q_1 and Q_2 are permutations, D a diagonal matrix with ± 1 on the diagonal and U a unit upper triangular gradient-type matrix. If we insert the expression for G_2^{-1} in μ we get

$$\mu(G_1 G_2^{-1}) = \mu(G_1 Q_2 U^{-1} D Q_1) = \mu(G_1 Q_2 U^{-1}).$$

Where the last equality holds, because μ is independent of permutations and diagonal scaling with ± 1. (This follows from the definition, but can be derived from Lemma 4.4.5 as well.) Note that $G_1 Q_2$ is a gradient-type matrix and that $0 \leq U^{-1}$ because of Lemma 4.4.2. This allows to apply Lemma 4.4.7 and we obtain

$$\mu(G_1 G_2^{-1}) = \mu(G_1 Q_2 U^{-1}) \leq \mu(U^{-1}) = 1.$$

□

We now have all the tools to prove the theorem.

Proof of Theorem 4.4.1. The goal is to obtain a bound on the size of the entries in $K^{(l)}$ in terms of the maximal entry of $K^{(0)}$. We start with observations that follow from the fact that $K^{(0)}$ is an $\mathscr{F}$-matrix.

(1) Because of Remark 4.3.6 we have $\mu(B^{(l)}) \leq \mu(B^{(0)}) \leq 1$ for all l. To determine the growth factor we only have to look at the size of the entries in $A^{(l)}$.
(2) In Theorem 4.3.3 we proved that for all l the matrix $A^{(l)}$ is positive definite. Gaussian elimination on positive definite matrices is unconditionally stable with growth factor $\rho < 1$, therefore the elimination of single V-nodes does not matter. Consequently we can focus on the effect of the elimination of the coupled V- and P-nodes.

Suppose that at a certain stage of Gaussian Elimination we eliminated k P-nodes, which means that we used k 2×2 pivots. Then we can reorder and split the original matrices

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},$$

such that the k 2×2 pivots are in the first blocks A_{11} and B_{11}. Note that with this choice B_{11} is square. If we insert the splitting of the matrices A and B in K we get after a little rearrangement

$$\left(\begin{array}{cc|cc} A_{11} & B_{11}^T & A_{12} & B_{21}^T \\ B_{11} & 0 & B_{12} & 0 \\ \hline A_{21} & B_{12}^T & A_{22} & B_{22}^T \\ B_{21} & 0 & B_{22} & 0 \end{array}\right).$$

The matrix B_{11} is square and invertible so we can interchange the first two block rows in the matrix:

$$\left(\begin{array}{cc|cc} B_{11} & 0 & B_{12} & 0 \\ A_{11} & B_{11}^T & A_{12} & B_{21}^T \\ \hline A_{21} & B_{12}^T & A_{22} & B_{22}^T \\ B_{21} & 0 & B_{22} & 0 \end{array}\right).$$

The Schur complement of the first two block rows is the same for both matrices. It is not influenced by the interchange of the rows. However, from the second matrix it is more easy to compute the Schur complement, it becomes

$$\tilde{K}^{(l)} = \begin{pmatrix} S_{22} & B_{22}^T - B_{12}^T B_{11}^{-T} B_{21}^T \\ B_{22} - B_{21} B_{11}^{-1} B_{12} & 0 \end{pmatrix}, \tag{4.15}$$

with

$$S_{22} = A_{22} - A_{21} B_{11}^{-1} B_{12} - B_{12}^T B_{11}^{-T} A_{12} + B_{12}^T B_{11}^{-T} A_{11} B_{11}^{-1} B_{12}.$$

Note that the matrix $\tilde{K}^{(l)}$ as given in (4.15) is not equal to $K^{(l)}$. Remember that $\tilde{K}^{(l)}$ is the Schur complement of $K^{(0)}$ after elimination of the k 2×2 pivots only. There can be a large set

of V-nodes that we need to eliminate before we get $K^{(l)}$. At least $K^{(l)}$ is a Schur complement of $\tilde{K}^{(l)}$. Because of observation (2) we know $\mu(K^{(l)}) \leq \mu(\tilde{K}^{(l)})$.

Now we can compute a bound for the growth factor as given in Equation (4.13):

$$\rho = \max_l \frac{\mu(K^{(l)})}{\mu(K^{(0)})} \leq \max_l \frac{\mu(\tilde{K}^{(l)})}{\mu(K^{(0)})} \leq \max_l \frac{\mu(S_{22})}{\mu(A)}, \tag{4.16}$$

where the last inequality holds because of observation (2). So a bound for $\mu(S_{22})$ gives a bound for the growth factor ρ.

$$\mu(S_{22}) \leq \mu(A_{22}) + \mu(A_{21}B_{11}^{-1}B_{12}) + \mu(B_{12}^T B_{11}^{-T} A_{12}) + \mu(B_{12}^T B_{11}^{-T} A_{11} B_{11}^{-1} B_{12}). \tag{4.17}$$

We now apply Lemma 4.4.5 a couple of times in order to get

$$\mu(S_{22}) \leq \mu(A_{22}) + 2\sigma(A_{12})\mu(A_{12})\mu(B_{12}^T B_{11}^{-T}) + k\sigma(A_{11})\mu(A_{11})\mu(B_{12}^T B_{11}^{-T})^2. \tag{4.18}$$

The matrices B_{12}^T and B_{11}^T are gradient-type matrices and B_{11}^T is invertible, so we can apply Lemma 4.4.8 and that gives $\mu(B_{12}^T B_{11}^{-T}) \leq 1$. This simplifies the bound to

$$\mu(S_{22}) \leq \mu(A_{22}) + 2\sigma(A_{12})\mu(A_{12}) + k\sigma(A_{11})\mu(A_{11}). \tag{4.19}$$

It is obvious that for all four subblocks of A we have $\mu(A_{ij}) \leq \mu(A)$. Furthermore A is positive definite, so it has strictly positive diagonal entries and the off-diagonal block A_{12} satisfies $\sigma(A_{12}) \leq \sigma(A) - 1 = \sigma - 1$. Using $\sigma(A_{11}) \leq \sigma$ and $k \leq m$ we finally get

$$\mu(S_{22}) \leq (1 + 2(\sigma - 1) + m\sigma)\mu(A).$$

If we use this expression in Equation (4.16) we finally get the bound for the growth factor. □

Compared to the general bound $\rho \leq (2.57)^{n+m-1}$ this bound is very sharp. Note that the result is not tied to the algorithm we proposed in Section 4.2 of this chapter. It holds for any feasible ordering. So it explains why nobody ever ran into trouble with stability for $\mathscr{F}$-matrices. See for example [80].

In practice even this bound is hardly attained. First of all because the bounds in Lemma 4.4.5 and Lemma 4.4.6 are quite pessimistic. Especially if we choose fill-reducing orderings for $A + B^T B$ the matrix $B_{12}^T B_{11}^{-T}$ won't be a matrix full of ones, but sparse, and therefore the last term in (4.17) will have a maximum element much smaller than $k\sigma(A)\mu(A)$. Even if $B_{12}^T B_{11}^{-T}$ would be a matrix full of ones it is likely that the product has a smaller maximum value, because in general summation of entries on a row in A_{11} will be much smaller than $\sigma(A)\mu(A)$.

For one special case we can give a slightly better bound, i.e. when A is diagonally dominant.

Theorem 4.4.9. *Let K be an $\mathscr{F}$-matrix with a feasible ordering. If submatrix A is diagonally dominant the stability bound for Gaussian elimination becomes*

$$\rho \leq 2m + 3.$$

Proof. We can give sharper bounds on the terms in (4.17). Because A is symmetric and diagonally dominant

$$\sum_{j=1, j\neq i}^{n} |a_{ij}| \leq |a_{ii}| \leq \mu(A).$$

It immediately follows that the off-diagonal blocks A_{12} and A_{21} have row sums smaller than $\mu(A)$. If we use that in Equation 4.14 we can derive a sharper bound for $\mu(A_{21}B_{11}^{-1}B_{12})$, i.e.

$$\mu(A_{21}B_{11}^{-1}B_{12}) = \mu(B_{12}^T B_{11}^{-T} A_{12}) \leq \mu(B_{12}^T B_{11}^T)\mu(A) \leq \mu(A).$$

We can apply Lemma 4.4.5 in combination with Lemma 4.4.6 to the last term in (4.17):

$$\mu(B_{12}^T B_{11}^{-T} A_{11} B_{11}^{-1} B_{12}) \leq k\mu(B_{12}^T B_{11}^{-T})\mu(B_{12}^T B_{11}^{-T} A_{11}) \leq 2k\mu(B_{12}^T B_{11}^{-T})^2 \mu(A_{11}) \leq 2m\mu(A).$$

Where we used again that $\mu(B_{12}^T B_{11}^{-T}) \leq 1$, $\mu(A_{ij}) \leq \mu(A)$ and $k \leq m$. Combining all estimates gives

$$\mu(S_{22}) \leq (2m+3)\mu(A).$$

Division by $\mu(A)$ provides the bound for the growth factor. □

Remark 4.4.10. The proofs of Theorem 4.4.1 and 4.4.9 can be extended to a wider class of saddle point matrices. We assumed K to be an $\mathscr{F}$-matrix, but we can weaken the demands. It is important to have A positive definite and $C = 0$, otherwise observation (2) does not hold anymore. In the proofs B^T being a gradient-type matrix is relevant for observation (1) and for the derivation of the bound $\mu(B_{12}^T B_{11}^{-T}) \leq 1$. However if B is not a gradient-type matrix, it might have other properties such that the entries of $B_{12}^T B_{11}^{-T}$ are bounded. That would be enough to compute a much better bound for the growth factor than the general one for indefinite matrices.

At the end of this section we show as an example the growth factor for $\mathscr{F}$-matrices from a Stokes problem on a square staggered grid. We will describe the problem in more detail in Section 4.6. We compute an ordering for the matrix with Algorithm 4.2.2 based on three different initial orderings for $F(A) + F(B^T)F(B)$ (natural, reverse Cuthill McKee [33] and approximate minimum degree [2, 3]). We perform Gaussian elimination on the reordered matrix, meanwhile monitoring the growth of the entries in the Schur complement. The results can be found in Figure 4.4 and Table 4.1.

In Figure 4.4 we plot $\rho^{(l)} = \mu(K^{(l)})/\mu(K^{(0)})$ for each Schur complement l for a square grid of size 9×9. The actual growth factor is the maximum of all $\rho^{(l)}$'s. For all three orderings the growth factor is rather small. The theoretical bound can be computed using Theorem 4.4.9 because the matrix is diagonally dominant. Here the bound is $\rho \leq 2m+3 = 163$, which is still quite far from the computed values of ρ. It is of course much better than the general bound $2.57^{223} \approx 2.60 \times 10^{91}$. Of interest is the effect of the different ordering algorithms on the growth of the elements. In case of the natural ordering the maximum growth is realized almost immediately after elimination of the first row in the grid. After that the growth remains constant except for a tiny peak half way. The best results in terms of a small growth factor are

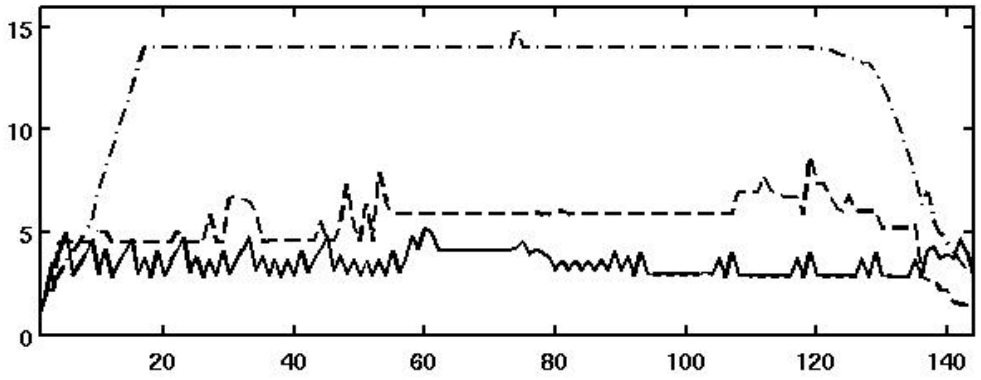


Figure 4.4: $\mu(K^{(l)})/\mu(K^{(0)})$ during Gaussian elimination for the natural (dash-dot), approximate minimum degree (dash) and reverse Cuthill McKee (solid) for the Stokes problem on a square grid of 9×9 cells, $n = 144, m = 80$.

matrix	n	m	nat	rcm	amd	bound
stokes3x3	12	8	6.0	5.6	6.5	19
stokes5x5	40	24	9.0	5.0	7.5	51
stokes9x9	144	80	15.0	5.0	8.5	163
stokes17x17	544	288	27.0	5.0	14.0	579
stokes33x33	2112	1088	51.0	5.0	15.0	2179
stokes63x63	8320	4224	99.0	5.0	15.0	8451

Table 4.1: Growth factors for several Stokes matrices for the natural ordering (nat), reverse Cuthill McKee (rcm), approximate minimum degree (amd) and the theoretical upper bound given by Theorem 4.4.9 (bound). The column n contains the number of V-nodes and m the number of P-nodes.

obtained for the reverse Cuthill McKee ordering. This is in agreement with a theorem in [8]. Apparently a small bandwidth is the best guarantee for a small growth factor.

To illustrate how the growth factor depends on the size of the grid and the number of P-nodes we show Table 4.1. As one can see the growth factor remains very small in case of reverse Cuthill McKee. It seems to be more or less independent of the grid size. This in contrast to the natural ordering that gives a growth factor that increases with the grid size, approximately a factor 1.5 if the number of nodes is doubled in two directions, more precisely the data is fitted exactly by $3(1+\sqrt{m+1})/2$. This is not really surprising because the number $\sqrt{m+1}$ is the number of grid cells in one dimension. Note that it still grows less than linear with m. The growth factor of approximate minimum degree is somewhere in between those two and appears to flatten for bigger problems. In all cases the bound provided by Theorem 4.4.9 is quite pessimistic.

We can conclude that for Gaussian elimination on $\mathscr{F}$-matrices the growth factor ρ is bounded by a number that grows linearly with m, the number of indefinite nodes. So Gaus-

sian elimination is numerically stable for $\mathscr{F}$-matrices and iterative refinement is hardly needed to compute accurate solutions.

4.5 Symbolic factoring

The application of Algorithm 4.2.2 requires the insertion of the P-nodes into the ordering for the V-nodes. In this section we show that this can be done very efficiently.

In Corollary 4.3.2 we expressed that the sequence of $B^{(l)}$'s is independent of A. We can exploit this in the insertion of P-nodes in the ordering for the V-nodes. The insertion can be done before the actual elimination based on the entries of B only. Especially if we deal with an $\mathscr{F}$-matrix it can be done very efficiently. We only need to know the fill pattern of B.

The P-nodes are inserted by Algorithm 4.5.1. The basic idea is that we track the elimination of P-nodes in a pointer array ptr. At the beginning all nodes have a pointer that points to the node itself, so $ptr(i) = i$ for all $i = 1,\ldots,m$. We add one extra element to this vector: $ptr(m+1) = m+1$. If a node v in V is coupled to two nodes p_1 and p_2 in P, it has to be eliminated together with one of them. Say we eliminate v and p_1 together. As we saw in the proof of Theorem 4.3.4 all V-nodes that were coupled to p_1 get coupled to p_2 instead. We reflect this by a change in the pointer array: $ptr(p_1) = p_2$. In the pointer array we can easily check what happened to the P-nodes by iteration on the pointer array until a fixed point is found. All fixed points are not yet eliminated P-nodes. There is freedom to choose p_1 or p_2. It is beneficial and in agreement with the minimum degree idea to pick the one with the least fill in the corresponding row of $B^{(l)}$. We will use an array nnz to store an estimate of the number of nonzeros in the rows of $B^{(l)}$. The array is used to make the decision whether p_1 or p_2 is to be eliminated. We will explain it in more detail in Remark 4.5.5.

Algorithm 4.5.1. *Insert the P-nodes in the ordering for V*

1. *Assume that we have an ordering $v_1, v_2, \ldots, v_n$ for the nodes in V.*
   Initialize $ptr(i) = i$ and $nnz(i)$ - the number of nonzeros on the i-th row in B - for all i.

2. *For $j = 1$ to n*

   (a) *Find p_1 and p_2, the column numbers of the entries on the v_j-th row of B^T.*
   If there is only one entry assign $p_2 := (m+1)$.
   If there are no entries at all assign $p_1 := (m+1)$ and $p_2 := (m+1)$.

   (b) *Follow pointers of p_1 and p_2 until fixed points are found:*
   repeat $p_1 := ptr(p_1)$ until $p_1 == ptr(p_1)$
   repeat $p_2 := ptr(p_2)$ until $p_2 == ptr(p_2)$

   (c) *If $p_1 == p_2$ do nothing.*

   (d) *If $(p_2 == (m+1))$ or $(nnz(p_1) \leq nnz(p_2))$ insert node p_1 (else p_2) immediately after v_j in the ordering to eliminate them using a 2×2 pivot. Change ptr and nnz to reflect elimination:*

$ptr(p_1) := p_2,$
$nnz(p_2) := nnz(p_1) + nnz(p_2) - 2.$

3. End

A few remarks on this algorithm.

Remark 4.5.2. One might fear cycles in pointer array *ptr* such that the algorithm never terminates step 2(a). Fortunately the only cycles that can appear in *ptr* are the fixed points. This can be simply explained by the following. Initially all points are fixed points. The vector is changed only at step 2(d) where one fixed point is replaced by a pointer to an existing other fixed point. So during the algorithm all paths in *ptr* will always end in a fixed point, hence the algorithm will certainly terminate.

Remark 4.5.3. The length of the iterations at step 2(b) is m in the worst case scenario. The number of iterations is related to the depth of the elimination tree with respect to the nodes in P. If we use a fill-reducing ordering it will hardly ever be bigger than $\sqrt{m}$. In any case the time used by the algorithm was always a fraction of the time used to compute the ordering for $A + B^T B$.

Remark 4.5.4. If we track the pointer array of two nodes p_1 and p_2, the fixed points they end on might very well be equal. If they are, the two entries have the same magnitude but opposite sign at the same position, so summing up gives zero: they annihilate. Hence in that case the node v_j is not coupled anymore to a pressure node, so we cannot insert any. This is conform step 2(c). Only if we have an $\mathscr{F}$-matrix we know exactly when this exact cancellation happens.

Remark 4.5.5. If there are two entries in a row, say at position p_1 and p_2, we have a choice in step 2(d) which of the two we will eliminate and insert in the ordering. Remember that the goal is to reduce the fill in the factorization. In Equation (4.9) we see that the amount of (new) fill in the Schur complement $A^{(l+1)}$ is determined by the number of nonzeros in the vector $\hat{\beta}/b$, that is the column in $B^{(l),T}$ that belongs to the P-node that is eliminated. So to reduce the number of nonzeros in $A^{(l+1)}$ we should eliminate the node with the fewest entries in its column. However it is a too expensive to compute the number of nonzeros in the columns of $B^{(l),T}$ precisely, therefore we estimate this number in the array *nnz*. At step 1 the number $nnz(i)$ is computed as the number of nonzeros on the i-th row in B. At that step it is still exact. However if at step 2(d) p_1 is eliminated we estimate that the number of nonzeros in the column of p_2 in $B^{(l+1),T}$ is equal to the sum of the number of entries in the two columns minus 2. We substract 2 because we delete the first row in $B^{(l),T}$. This number is an overestimate, because it does not take in account exact cancellation. Nevertheless the approximation is good enough to make the right decision in step 2(d).

If we do not have an $\mathscr{F}$-matrix, a similar algorithm can be used. Corollary 4.3.2 holds for general saddle point matrices with $C = 0$, so still only B and an ordering for V are needed to insert the P-nodes. Nevertheless it will be less efficient because we cannot benefit from a simple structure of B. Furthermore we need to monitor the size of the entries in $B^{(l)}$ as well and

it will be more difficult to detect exact cancellation. Anyway the insertion cannot be done in the symbolic factoring phase. We need the size of the entries in B and we have to do calculations with reals instead of integers.

4.6 Numerical results

In this section we will show the results of our algorithm for two sets of matrices. We implemented Algorithms 4.2.2 and 4.5.1 in MATLAB 7.1.0.183 (R14) Service Pack 3. The first ordering is computed with MATLAB's symmetric approximate minimum degree ordering SYMAMD [2, 3].

We compare the results of Algorithm 4.2.2 to that of PARDISO version 3.1 (serial) [66, 68, 67] that is able to factorize indefinite symmetric matrices. It uses either AMD (approximate minimum degree) or METIS (nested dissection, [45]) as basic ordering. We use the standard parameter settings except that we follow the advise of the manual to use scaling (IPARM(11) = 1) and weighted matchings (IPARM(13) = 1) in case of highly indefinite matrices like saddle point problems. If we choose to switch off weighted matchings, PARDISO is still able to build a factorization, but the error in the solution without using iterative refinement raises from $\mathscr{O}(10^{-13})$ to $\mathscr{O}(10^{-6})$. So we really need weighted matchings to compute an accurate factorization.

The package PARDISO offers the possibility to ignore its ordering algorithms and instead perform factorization based on an ordering provided by the user. We use this facility with the ordering of Algorithm 4.2.2 as input. In the tables in this section, we will use "PARDISO(amd/metis/uo)" to denote the PARDISO factorization based on amd, metis and the user ordering respectively. The MATLAB factorization is called "LDL^T(amd)".

All numerical experiments were done on a PC with two 2.4 GHz AMD Opteron processors and 7.6 GB memory.

4.6.1 Stokes flow in a driven cavity

The first problem is a two-dimensional Stokes equation in a driven cavity. Here the following set of equations has to be solved on the unit square Ω

$$\left.\begin{array}{rcl} -\nu\Delta u + \nabla p &=& 0, \\ \nabla \cdot u &=& 0. \end{array}\right\} \tag{4.20}$$

where $u(x,y)$ is the velocity field and $p(x,y)$ the pressure field; the parameter ν controls the amount of viscosity. The non-trivial solution is determined by the boundary conditions that are zero on three sides of the unit square. At the upper boundary ($y = 1$) we prescribe a horizontal velocity $u(x,1) = 1$.

We can get rid of the parameter ν by defining a new pressure variable $\bar{p} = p/\nu$. If the first equation is divided by ν, we can substitute p by $\bar{p}$ and the parameter ν is gone. So we may assume that $\nu = 1$.

matrix	n	m	$n+m$	nnz
stokes3x3	12	8	20	48
stokes5x5	40	24	64	180
stokes9x9	144	80	224	684
stokes17x17	544	288	832	2,652
stokes33x33	2,112	1,088	3,200	10,428
stokes65x65	8,320	4,224	12,544	41,340
stokes129x129	33,024	16,640	49,664	164,604
stokes257x257	131,584	66,048	197,733	656,892
stokes513x513	525,312	263,168	788,480	2,624,508

Table 4.2: The set of Stokes matrices. n is the number of V-nodes, m the number of P-nodes and nnz the number of nonzeros in the upper triangular part of the matrix.

If the equations are discretized on a uniform staggered grid (a C-grid) with mesh size h we get an $\mathscr{F}$-matrix. It is singular because the pressurefield is determined up to a constant. We get rid of this problem by fixing one pressure node in the domain. So the number of pressure nodes is reduced with one. Table 4.2 shows the size and the number of nonzeros in the upper triangular part of the Stokes matrices.

The results of factorization of these matrices with different ordering algorithms can be found in Table 4.3. Obviously Algorithm 4.2.2 is able to produce an ordering that gives a factorization with low fill. For all grid sizes it results in a factorization that is sparser than the factorizations of PARDISO. The large gap between the numbers of nonzeros in the last two columns is quite remarkable, because the very same ordering is used. The user ordering (UO) we give to PARDISO is the one computed in matlab with Algorithm 4.2.2. We use that ordering as well to compute the LDL^T(amd) factorization in matlab. The factorizations should be the same. The only explanation we have is that probably the L factor of PARDISO contains many entries close to machine precision, because it cannot detect exact cancellation, as we can in our algorithm.

4.6.2 Matrices of Tůma

The second set of testmatrices consists of seven matrices provided by Miroslav Tůma of which four can be found in [80]. In Table 4.4 we show the size and number of nonzeros of the matrices. In [80] the matrices have a slightly larger number of nonzeros because the sparse matrix format contains some 'zero nonzeros' that we removed *a priori*. We have to remark that the Tůma-matrices do not satisfy the assumption that the number of nonzeros in a row in B is smaller than the number of nonzeros in A. All A's in the Tůma set have at most three nonzeros per row, while at least a quarter of the rows in B has more than three nonzeros. Maybe Condition 4.2.1 is not the best in that case. Possibly a weakening of the condition could provide

matrix	PARDISO (amd)	PARDISO (nd)	PARDISO (uo)	LDL^T (amd)
stokes3x3	146	129	114	82
stokes5x5	646	743	555	403
stokes9x9	3,203	3,668	2,829	2,134
stokes17x17	18,060	20,731	15,296	11,415
stokes33x33	98,936	114,307	79,131	63,304
stokes65x65	529,368	594,132	448,320	365,311
stokes129x129	2,761,774	3,014,234	2,444,302	2,039,458
stokes257x257	13,999,517	14,615,960	12,682,925	10,877,966
stokes513x513	69,350,006	69,354,497	63,766,827	55,900,331

Table 4.3: Number of nonzeros for various factorizations of the Stokes matrices.

matrix	n	m	$n+m$	nnz
S3P	270	207	477	1,008
M3P	2,160	1,584	3,744	8,136
DORT2	7,515	5,477	12,992	28,440
DORT	13,360	9,607	22,967	50,560
L3P	17,280	12,384	29,664	65,376
dan2	63,750	46,661	110,411	318,304
novak	152,645	114,113	266,758	744,912

Table 4.4: The set of Tůma's matrices. n is the number of V-nodes, m the number of P-nodes, nnz the number of nonzeros in the upper triangular part of the matrix.

a further improvement of the performance.

Table 4.5 contains the number of nonzeros of the factors with different fill-reducing ordering algorithms. We compare the results of Algorithm 4.2.2 with the results of Tůma and PARDISO.

The ordering of Tůma is better than the two orderings of PARDISO. However in all cases our ordering gives the sparsest factors. The big difference between the approximate minimum degree and nested dissection ordering in PARDISO is quite remarkable. It must be due to the structure of the Tůma matrices, because for the Stokes matrices the results for both orderings are similar. The qualitative difference between the two sets of matrices is noticed as well in the effect of the choice we made in Remark 4.5.5. If we do not eliminate the node that has the least nonzeros in its column of B^T, but simply pick the first node, the results of LDL^T(amd) in Table 4.3 hardly change, whereas the results in Table 4.5 seriously deteriorate.

matrix	PARDISO (amd)	PARDISO (nd)	PARDISO (uo)	LDL^T (amd)	Tůma
S3P	3,154	3,290	2,754	2,195	2,957
M3P	60,305	61,144	45,248	35,553	44,002
DORT2	558,522	276,376	243,088	208,393	231,312
DORT	1,278,505	641,788	598,130	527,602	551,215
L3P	1,445,900	1,000,789	785,689	690,932	-
dan2	4,465,051	3,040,497	2,686,839	2,240,202	-
novak	15,453,022	9,785,185	9,351,503	8,217,979	-

Table 4.5: Number of nonzeros in various factorizations of the Tůma matrices. The column Tůma contains results from [80].

4.7 Discussion

In this chapter we proposed a new algorithm to compute a stable fill-reducing ordering for $\mathscr{F}$-matrices, a special class of symmetric saddle point matrices. The algorithm is based on a simple idea: compute an ordering for the V-nodes first and then add the P-nodes. The ordering for the V-nodes is a fill-reducing ordering for $A+B^TB$. The P-nodes are added under the condition "eliminate P-nodes as soon as possible". The final ordering is guaranteed to be feasible and in case of $\mathscr{F}$-matrices it can be computed very fast using symbolic factorization.

In Section 4.3 we showed that the ordering guarantees that much of the structure of the saddle point problem is preserved in the Schur complements during Gaussian elimination, in particular $C^{(l)}$ remains empty and $B^{(l)}$ is independent of $A^{(l)}$. Positive definiteness of A and the $\mathscr{F}$-matrix-properties are also preserved. From an engineering view this is a very attractive property.

One of the important results is Theorem 4.4.1 where we give a bound for the growth factor in Gaussian elimination on $\mathscr{F}$-matrices. It explains why in the literature on the factorization of saddle point matrices numerical stability is not a big issue. The general bound for the growth factor grows exponentially fast, but it is too pessimistic. At least in case of $\mathscr{F}$-matrices the growth factor is bounded by a number that grows linearly with the dimension. It is likely that this holds for a larger class of saddle point matrices, because it seems possible to weaken the assumptions of the proof. If B is not a gradient-type matrix it might have other properties such that $B_{11}^{-T}B_{12}$ is bounded, as we have argued in Remark 4.4.10.

The numerical experiments on the Stokes and Tůma matrices show that the algorithm is able to produce a good factorization with a sparser structure than the factorizations of other methods.

Another nice property is that the very same algorithm can be used for more general $\mathscr{F}$-matrices like the 2D Navier-Stokes equations (with Coriolis force) on a C-grid.

There is a lot more to study. In some cases Condition 4.2.1 should be weakened to get a sparser factorization. There might be better ways to compute the first ordering, because we

loose information in the construction of $F(A)+F(B^T)F(B)$. For example, the last matrix will be dense if B contains a full row. One more important question: how to generalize the algorithm such that it is able to handle a saddle point matrix that is not an $\mathscr{F}$-matrix? Clearly there are several ways to do this, but which one is the best, and can we keep the nice properties we showed in Section 4.3?

Finally, in this chapter we showed that the simple ideas behind the algorithm make sense. The construction of the ordering is straightforward and fast. It keeps nice properties of the matrix during Gaussian elimination and in the numerical experiments it appears to be powerful enough to result in a factorization with significantly lower fill than the factorizations of existing methods.

Acknowledgement

We thank Miroslav Tůma for the permission to use his set of $\mathscr{F}$-matrices from [80].

Chapter 5

A fill-reducing ordering for use in incomplete LU factorizations

5.1 Introduction

In Chapter 3 we described the two methods to solve a linear system $Ax = b$, with $A \in \mathbb{R}^{n \times n}$ sparse and $x, b \in \mathbb{R}^n$: we can choose between direct and iterative methods. In general, these approaches are quite different, however, in this chapter we will combine elements of both.

Direct methods compute an exact factorization of the matrix, hence, $A = LU$ with L a lower triangular matrix and U an upper triangular matrix. Herewith the linear system can easily be solved using forward and backward substitution. The amount of work depends on the amount of non-zeros (the fill) in the factors L and U. Smart ordering of the variables in x can reduce the fill substantially and it is attractive to incorporate this idea in incomplete factorizations.

On the other hand there are the iterative methods. These methods generate a sequence $\{x_1, x_2, ..., x_n\}$ of approximations to the solution of $Ax = b$; popular are the Krylov subspace methods. Preconditioning, e.g. by incomplete LU factorization, can improve the performance of the methods considerably. Thus constructed iterative methods appear to be competitive to direct ones for large linear systems.

An historical overview of iterative methods can be found in [65]. In the conclusion of that paper it is remarked that ideas from direct and iterative methods should be combined to solve problems that are hard to solve by iterative methods.

In this chapter we explore the possibilities to use orderings from direct methods in incomplete LU factorizations. The idea of combining both worlds is not completely new. In [24] a huge number of orderings is combined with a ICCG(0) factorization in preconditioned Conjugate Gradient method and in [16] the effect of ordering in ILU factorizations with a high level of fill is considered.

In the next two sections we will shortly introduce the techniques from the two worlds that we will use: first the fill-reducing ordering as used in direct methods and thereafter the incomplete LU factorization as preconditioner in iterative methods. The most important and most

ordering	fill ($\times 10^6$)	relative fill	construction time (sec)	solve time (sec)
natural ordering	33.1	102.1	34.44	1.02
reverse Cuthill-McKee (SYMRCM)	22.2	68.6	18.74	0.72
minimum degree (SYMMMD)	3.7	11.5	3.00	0.21
approximate minimum degree (SYMAMD)	3.6	11.1	2.66	0.22

Table 5.1: Number of nonzeros (fill) in the factors of a Poisson matrix on a 255×255 grid for several orderings. Between brackets: the corresponding MATLAB functions.

extensive section of this chapter is Section 5.4. There we will explain in detail how to combine both techniques in a new algorithm for ordered incomplete LU factorization. In Section 5.5 the algorithm is applied to a few problems and the chapter closes with some conclusions in Section 5.6.

5.2 Fill-reducing orderings in direct methods

In the construction of an exact LU factorization the ordering of the variables in the process of Gaussian elimination is very important. A smart ordering can reduce the fill (the number of non-zeros) in the factors. Fill reduction has several advantages: less memory to store the factors is needed, computations with the factors become cheaper and the construction of the factors becomes cheaper as well.

Because of these advantages one would like to have the special ordering that generates the least fill. Unfortunately the minimum fill problem is NP-complete [89], so it is not practical to compute that ordering. This is not a big issue, because fortunately there are efficient heuristic algorithms that produce a near-minimum fill ordering. In [53, Ch.3] several of these algorithms for ordering in sparse linear systems are presented. We introduced the two most important in Section 3.1

In this chapter we will avoid going into further details, instead we will show the importance of an appropriate ordering by an example. In Table 5.1 we applied four different ordering algorithms to a Poisson matrix on a square grid of 255×255 nodes. We computed the fill and the relative fill, i.e. the fill of the factors divided by the fill of the original matrix. Furthermore we measured the construction time and the time needed to solve an equation with the factors. We did the computations in MATLAB and the ordering functions used are mentioned in the table in brackets. The table not only shows that fill reduction reduces the amount of work in the solution phase, but it does even more seriously in the construction phase.

There is one important disadvantage in the use of direct methods: even if the best fill-reducing ordering is used, the fill in the factors grows with the problem size. If we increase the problem size the fill in the factorization will have a seriously larger increase. Therefore, with direct methods the memory limits of any computer are easily reached.

5.3 Preconditioning via incomplete LU factorization

The general idea of preconditioning is to construct a matrix $\hat{A}$, that is a good approximation for A, but such that equations with $\hat{A}$ are much easier to solve than equations with A. With a good approximation we mean that the eigenvalues of $\hat{A}^{-1}A$ are not far from one.

One way to obtain a preconditioner is the construction of an incomplete LU factorization. A lower triangular matrix $\hat{L}$ and an upper triangular matrix $\hat{U}$ are computed such that $\hat{A} = \hat{L}\hat{U}$ is close to A. The incomplete factors $\hat{L}$ and $\hat{U}$ should be sparser than the exact factors in any fill-reducing ordering. The sparser the incomplete factors, the cheaper we can solve equations with $\hat{A}$.

The general framework for the construction of an incomplete LU factorization is the following. First of all a subset of variables $x_1 \subset x$ is selected; this set may consist of a single variable. Let $x_2 = x \setminus x_1$ be the complement of this set. The matrix A is reordered according to this partitioning and split into four parts,

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}. \tag{5.1}$$

The next steps are the approximation of A_{ij} by matrices $\hat{A}_{ij}$ and the computation of the approximate Schur complement $S = \hat{A}_{22} - \hat{A}_{21}\hat{A}_{11}^{-1}\hat{A}_{12}$. In most ILU algorithms the approximation will be such, that the Schur complement is easy to compute. This is the case if $\hat{A}_{11}$ is a diagonal matrix. Now we have the following incomplete factorization for A,

$$\begin{bmatrix} I & 0 \\ \hat{A}_{21}\hat{A}_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} \hat{A}_{11} & \hat{A}_{12} \\ 0 & S \end{bmatrix}. \tag{5.2}$$

There are many variants of incomplete LU factorization. One can choose for example between a drop-by-position or a drop-by-size criterion, whether to apply Gustafsson-modification, design a sophisticated search algorithm to select the right x_1, and so on. For an overview of preconditioning methods in general and incomplete LU factorizations in particular we refer to [65] and [53, Ch.8].

The use of Krylov subspace methods in combination with preconditioning via incomplete LU factorization is a powerful tool as is shown in [10] and [9], where MRILU is used.

5.4 Fill-reducing orderings in incomplete LU factorization

In this section we will propose a method to combine fill-reducing orderings and incomplete factorization. The central idea is quite simple: compute some fill-reducing ordering for the matrix and start performing Gaussian elimination on the matrix according to this ordering, interrupt the process a number of times for a search for a set of nodes that can be eliminated inexactly (see Algorithm 5.4.1).

Algorithm 5.4.1. *Ordered incomplete LU-factorization.*

1. *Compute a fill-reducing ordering for the matrix.*
2. *Alternate the following steps:*
   (a) *Reduce the matrix by exact elimination according to the ordering.*
   (b) *Reduce the matrix by inexact elimination.*

In Section 3.1 we explained that Gaussian elimination on a matrix is related to pruning of the elimination tree. The pruning at step 2(a) of the algorithm above is regular, so we cut off a number of leaves of the elimination tree. Clearly this does not affect the structure of the tree. It loses some of the leaves and might become less deep, but the nodes that remain keep the same parents.

The pruning at step 2(b) is totally different. The nodes that are eliminated here can be spread all over the tree. Pruning in this case means that we cut out nodes in the middle of a branch. Even the root is candidate for pruning. Obviously this pruning will change the tree: nodes lose their parents and get new ones somewhere in the tree; whole branches will move. The inexact elimination has to be done very carefully, otherwise we totally ruin the tree and we make the effort we made in step 1, where we construct the fill reducing ordering and the corresponding tree, worthless.

As an example we will apply the Algorithm 5.4.1 to the matrix in Equation (3.4). Suppose that the fill reducing ordering computed in step 1 is the trivial fundamental ordering $q_F = \{1,2,3,4,5,6,7,8,9\}$. The elimination tree that corresponds to this ordering is plotted in Figure 3.1. A possible evaluation at step 2 could be the following: 2(a) the nodes $\{1,2\}$ are eliminated exactly, 2(b) the nodes $\{5,7\}$ are eliminated inexactly, 2(a) nodes $\{3,4\}$ exactly, 2(b) node $\{8\}$ inexactly, finally 2(a) nodes $\{6,9\}$ exactly. In fact we changed the ordering to $q' = \{1,2,5,7,3,4,8,6,9\}$, where some eliminations won't be carried out exactly.

In the second half of this section we will pay attention to the conditions for inexact elimination. First we want to focus on the question when to apply steps 2(a) and 2(b). For an answer to that question we have to study the ordering of the elimination tree.

5.4.1 Tree ordering

In step 1 of Algorithm 5.4.1 we compute an ordering that reduces the fill in the factors of an exact factorizations. We will call this ordering the *fundamental ordering*. Given this ordering there is some freedom to change it without changing the structure of the elimination tree, which determines the fill in the factors. This is caused by the fact that we are free to renumber the nodes in the elimination tree, as long as parents have higher numbers than the children. Orderings that give elimination trees with the same structure are called *equivalent*.

Matrix (3.4) with the trivial fundamental ordering $q_F = \{1,2,3,4,5,6,7,8,9\}$ gives the elimination tree in Figure 3.1. However the nodes 1 and 2 are both leaves, so they can be

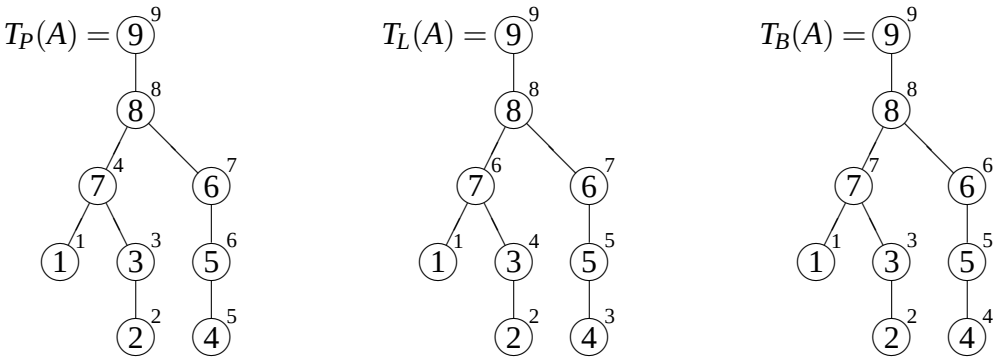


Figure 5.1: Elimination tree (from the example in Section 3.1) with postordering (T_P), level ordering (T_L) and bare-branch ordering (T_B). The numbers outside the circles denote the position of the node in the elimination order.

interchanged without changing the structure of the elimination tree. Consequently the ordering $q'_F = \{2,1,3,4,5,6,7,8,9\}$ is equivalent to q_F.

An elimination tree generally has many equivalent orderings. However most algorithms prefer the *postordering* for the elimination tree, which numbers all the subtrees consecutively. For our example the elimination tree with postordering is the left tree in Figure 5.1. The subtree rooted at node 7 is numbered first. The postordering permutation equivalent to the trivial ordering is $q_P = \{1,2,3,7,4,5,6,8,9\}$.

In Algorithm 5.4.1 we want to jump a few times between exact elimination (step 2(a)) and inexact elimination (step 2(b)). In that case the postordering for the elimination tree is not very convenient. The problem is that during the exact elimination all changes happen on a small subtree, while during the inexact elimination changes appear all over the tree. In the example one can see that the elimination of the first three nodes in the postordering ($\{1,2,3\}$) does not affect any of the nodes in the subtree rooted at node 6. The point is that we would like to balance the pruning of step 2(a) better over the whole tree, because in that case the changes in the underlying adjacency graph are well spread over the graph. Then a new search in step 2(b) for a set that can be eliminated inexactly is really different from the previous search.

We would like to have an ordering of the elimination tree, such that elimination of a number of nodes changes the dependencies of the remaining nodes in as many subtrees as possible. This obviously happens if all leaves are eliminated. The leaves form an independent set, so they can be eliminated simultaneously. If the leaves are removed from the tree, we get a new tree, that of course has a new set of leaves. That set can be eliminated as the next level. In this way we get an ordering that we will call a *level ordering* where all leaves are numbered first, then the leaves of the reduced tree, etcetera. Furthermore we define a *level* as the set of all leaves of a (reduced) tree. The middle tree in Figure 5.1 has a level ordering. There are five levels, namely the sets of nodes $\{1,2,4\}$, $\{3,5\}$, $\{7,6\}$, $\{8\}$ and $\{9\}$, which gives the equivalent ordering $q_L = \{1,2,4,3,5,7,6,8,9\}$.

The attempt to reduce the system via inexact elimination could be naturally placed between

the regular exact elimination of two consecutive levels. Unfortunately in the level ordering the number of levels is equal to the depth of the tree, which becomes a large number for large problems. This means that we have to perform the relative expensive procedure of a search for inexact elimination many times. Furthermore the last levels exist of very few nodes. Exact elimination of these levels change only a few dependencies. A new attempt for inexact elimination seems vain in advance.

This problem can be tackled by merging consecutive levels, but we prefer a more sophisticated approach. To reduce the number of levels we will define a new type of equivalent ordering. The elimination tree contains many chains of parents with only one child. In general close to the root the chains are long, whereas close to the leaves the chains are short. We use the idea of chains in a variant of the level ordering in order to reduce the number of levels. In the *bare-branch ordering*, we will number chains instead of leaves. In this new kind of equivalent ordering we start again at the leaves. We number a leaf and if this leaf is an only child we number the parent too. All only-child ancestors are numbered until we find an ancestor with more then one child. We do this for all leaves and put them with their only-child-ancestors in the first level. The second level exists of the leaves and only-child-ancestors of the reduced elimination tree and so on. The chains of parents with only one child can be seen as bare branches of the elimination tree, which explains the name of the ordering. Figure 5.1 shows a bare-branch ordering. The tree has only two levels, namely the sets $\{1,2,3,4,5,6\}$ and $\{7,8,9,10\}$. Consequently the bare-branch ordering is $q_B = \{1,2,3,4,5,6,7,8,9\}$ and we see that the fundamental ordering is more than equivalent to q_B, it is exactly the same.

In the bare-branch ordering the number of levels no longer depends on the depth of the tree, but on its width. The width of a tree is equal to the number of leaves, which is related to the number of splittings. A splitting is a node of the elimination tree that has at least two children. The number of splittings determines the number of levels in the bare-branch ordering. In a balanced tree the splittings will be equally spread over the subtrees and the number of levels will be approximately $\log_2(\#\text{leaves})$. In fact this number is an overestimate for the number of levels, because splittings can have more than two branches and probably the tree is not perfectly balanced, which also reduces the number of levels.

For the example trees in Figure 5.1 the differences between the orderings are small. However if we want to apply Algorithm 5.4.1 to large matrices it is important to use the bare-branch ordering. The advantage over the postordering is that the bare branch orderings ensures that in exact elimination the pruning is well balanced over the tree. In one level all bare-branches are eliminated at the same time. The advantage over the level ordering is that we have a small number of levels even if we have a large matrix. Numerical experiments support this. In Table 5.2 we compare the number of levels of the bare-branch ordering with that of the level ordering. The matrix is a Poisson matrix for which an approximate minimum degree ordering is constructed with the MATLAB-function SYMAMD. We also added a column with the estimate $\lceil \log_2(\#\text{leaves}) \rceil$, which appears to be almost equal to the number of levels in the bare-branch ordering.

We note that it is easy to transform a postordering into one of the other orderings; only one sweep through the elimination tree is required. Hence the bare-branch ordering can be

grid size	number of nonzeros	tree depth	tree width	number of levels level	bare-branch	est
3×3	33	6	4	6	2	2
7×7	217	19	13	19	3	4
15×15	1065	50	91	50	5	7
31×31	4681	115	443	115	7	9
63×63	19593	315	1915	315	8	11
127×127	80137	678	7931	678	11	13
255×255	324105	1484	32251	1484	12	15
511×511	1303561	3299	130043	3299	14	17
1023×1023	5228553	6349	522235	6349	16	19
2047×2047	20942857	14730	2093051	14730	18	21

Table 5.2: Number of levels in two equivalent orderings for the Poisson matrix on several grid-sizes. The fundamental ordering is approximate minimum degree. Shown are the level ordering (level) and the bare-branch ordering (branch). The last column (est) contains the estimate for the number of levels based on the width of the tree.

constructed in linear time.

5.4.2 Inexact elimination

Now we have introduced the bare-branch ordering it is natural to apply step 2(b) in Algorithm 5.4.1 directly after the exact elimination of a level in step 2(a). Then we try to reduce the size of the matrix by inexact elimination between the exact elimination of two consecutive levels. As the number of levels is relative small, we don't have to apply step 2(b) too often.

As we stated before, the inexact elimination of nodes should be done carefully. The elimination of nodes that are not a leave, will change the tree. For example, if we remove node 7 in a tree in Figure 5.1, the tree will definitely change, e.g. due to exact elimination of node 7 in the adjacency graph in Figure 3.1 a new edge is created between the nodes 1 and 2. The nodes 1 and 3 lose 7 as their parent and get a new one somewhere else in the tree: 2 becomes the parent of 1 and 8 the parent of 3. The structure of the tree is lost, because leave 1 moves to a different branch. As we want to maintain the structure of the tree as much as possible, we demand that the new parent is one of the ancestors of node 7: node 8 or 9. Hence the edge between nodes 1 and 2 in the adjacency graph, which causes 2 to become the parent of 1, is unwanted. In general the nodes in disjoint subtrees are not connected in the adjacency graph and are never allowed to become connected by inexact elimination. If such edges are created by the elimination of a node in step 2(b), we have to drop them immediately.

This demand has some consequences for the framework of incomplete LU factorization as sketched in Section 5.3. Suppose we have chosen the set $x_1 \subset x$, computed $x_2 = x \setminus x_1$ and partitioned the matrix A as in Equation 5.1. We will use a diagonal approximation for A_{11}, so $\hat{A}_{11} = D_{11}$, but the blocks A_{12} and A_{21} will not be approximated. Now we can compute the

Schur-complement $S = A_{22} - A_{21}D_{11}^{-1}A_{12}$. The matrix S has probably fill, where A_{22} has not. Hence we drop all new entries and approximate S with $\hat{S}$, that has the same fill pattern as A_{22}. Of course the more fill is dropped the worse the approximation will be. So we have to look for a set such that the dropping in both A_{11} and the Schur-complement is limited. This is what the algorithm does: look for a set $x_1 \subset x$ such that the $E_{11} = A_{11} - D_{11}$ and $E_{22} = S - \hat{S}$ is "relatively" small. In the current algorithm this dropping is computed exactly, which of course is expensive. Smart estimates might make the algorithm much faster, but because at the moment we are mainly interested in the possibility at all we won't pay attention to that in this chapter.

The procedure sketched here will end up in the following incomplete LU factorization.

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_{21}D_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} D_{11} & A_{12} \\ 0 & \hat{S} \end{bmatrix} + \begin{bmatrix} E_{11} & 0 \\ 0 & E_{22} \end{bmatrix}. \tag{5.3}$$

In general this incomplete factorization will be a good one if the error matrix E is small relative to A. Of course we cannot compute $A^{-1}E$, so we look at $D^{-1}E$, where D is the diagonal of A. We can do this without much concern, because we still relate the error to the size of the entries in A. In the search algorithm we look for a set $x_1 \subset x$ such that $D^{-1}E$ is small in some norm. The existence of such a set depends on the drop tolerance and the type of problem. If a nonempty x_1 is found, this set is eliminated in the way sketched above.

Because the fill patterns of $\hat{S}$ and A_{22} are the same, we ensure that no entries are created between disjoint subtrees. If we remove the eliminated nodes from the tree, connecting orphans to the first ancestor of the eliminated parent, we get a new tree, that is an elimination tree for $\hat{S}$. The corresponding ordering is still fill reducing, so the original exact factorization procedure can be continued.

Applying the algorithm sketched so far we observe that for large values of the drop tolerance the elimination tree falls apart. After a few times of exact and inexact elimination we get a forrest instead of a single tree. This is caused by the fact that the Schur complement becomes reducible or in graph terminology the grid is no longer connected: it consists of two or more components. With the tree falling apart the number of iterations grows seriously and zeros appear on the diagonal of U, so the factorization becomes singular. Of course these zeros can be easily replaced by a nonzero number, but it will still have a bad effect on the number of iterations.

This motivates to put an extra demand on the selection of the sets, namely $x_2 = x \setminus x_1$ should be connected. Unfortunately it is not easy to check whether the deletion of a node will create two or more components. Therefore we make an even stronger demand: a node is considered for inexact elimination only if its neighbours in x_2 are connected. This is easy to check because the involved grid is rather small. For a five point stencil the demand will prevent any inexact elimination on the first level, so at that level we relax the demand. The neighbours of the neighbours in x_2 should be connected.

With this extra restriction the method becomes quite robust. For any drop tolerance the factorization is non singular. For the problems we treat in the next section this restriction has

hardly any effect on the fill, but the number of iterations reduces with a factor two or more.

The order of the nodes in the search for the set x_1 appears to have some influence on the factorization also. The best search order seems to be the opposite of the exact elimination order. The nodes in the highest level are considered for inexact elimination first. This order will have the largest effect on the fill of the factors.

5.4.3 Implementation

The algorithm sketched in this section has been implemented in MATLAB. The current version of the algorithm is not very efficient. This is mainly caused by the fact that in the search for incomplete elimination the errors are computed exactly. However in this chapter we are mainly interested in the possibility at all of combining fill-reducing orderings and incomplete factorization. We expect that smart estimates for the error and a detailed time study of the algorithm can make it competitive to the exact factorization.

The algorithm requires a fill-reducing ordering with a balanced elimination tree and a large width. This means that methods like minimum degree or nested dissection are appropriate. The reversed Cuthill McKee ordering reduces the bandwidth of the matrix, which means that often the first off-diagonals are completely filled. A tridiagonal matrix has small bandwidth, but its tree is not interesting: it exists of a single chain of nodes. As the reversed Cuthill McKee ordering gives an extremely unbalanced tree, we cannot use it in Algorithm 5.4.1.

5.5 Numerical results

In this section we will show numerical results for three problems: a Poisson equation, a convection-diffusion equation and a Stokes equation. In all cases we apply a priori a (block) diagonal scaling. We use the approximate minimum degree ordering that is available in MATLAB and in the incomplete factorization we use Gustafsson-modification, i.e. we lump dropped elements on the diagonal. We will focus on the fill and the quality of the factorization. The quality is measured by the number of iterations in a Krylov subspace method that is needed to reach an accuracy of 10^{-6}.

We will compare our method to the incomplete LU or incomplete Cholesky factorization of MATLAB with natural and approximate minimum degree ordering.

All experiments are run on a PC with 2 GB memory and a 3 GHz processor.

5.5.1 Laplace equation

First the method is tested on the Laplace equation on the unit square with Dirichlet conditions $u = 1$ on the boundary. Hence, the solution is simply the constant value 1. We use a standard

five-point discretization on a uniform grid. So we have the equations

$$\begin{aligned} \Delta u &= 0 \text{ in } \Omega, \\ u &= 1 \text{ on } \partial\Omega. \end{aligned}$$

For the starting vector in the Krylov subspace method (here PCG) we disturb the solution. The distortion exists of a single hump, a smooth function. The starting vector is

$$u_0(x,y) = 1 + (xy(1-x)(1-y))^2 e^{x^2 y}.$$

The problem is equivalent to the "Uniform-2D" problem in [9]; the initial error is exactly the same. In Table 5.3 it is shown how the fill and the number of iterations behave if the problem size increases for various values of the dropping parameter ε. The problem size is just the square of the number of grid points in x- and y-direction and the fill is the average number of non zeros in L+U per row. The matrix A itself has approximately 5 nonzeros per row. In the first column one observes how the fill increases using the minimum degree ordering without dropping. So for the largest problem the fill of the preconditioner is more than 11 times as much as that of the original matrix. In the other columns one observes also an increase of the fill but it becomes less pronounced with increasing ε (look for instance to the ratio of the fill for the largest and of that for the smallest problem). Of course, for the exact elimination the number of iterations is precisely one. For the case $\varepsilon = 0.01$ the fill is still rather high resulting in an accurate factorization which amounts to only 2 iterations to gain 6 digits. The case $\varepsilon = 0.2$ has a more attractive fill for practical purposes whereas the number of iterations is still rather low. Because of the demand of connectivity the algorithm cannot generate a factorization with less fill than 6.9 in the last column of Table 5.3. A further reduction of the fill would cause the tree to fall apart, which is not allowed.

In Table 5.4 the number of iterations and the number of nonzeros per row in $L+U$ are shown for the incomplete Choleski factorization with natural and approximate minimum degree ordering. The drop tolerances are chosen such that the level of fill is more or less the same as that in the last row of Table 5.3. Comparing the results we conclude that the factorization of OILU (an acronym for Algorithm 5.4.1) is better than an ordinary incomplete LU factorization with natural ordering as long as we want a fill that is more than one and a half times the fill of the original matrix.

The amount of work per unknown to solve the system during the iteration is roughly the product of the fill and the number of iterations where the fill is simply 5 times the relative fill. So for the largest problem the amount of work is respectively 55, 86, 105, 84, 95, 157, 220, 255, 323 and 669 with increasing ε.

The last two values are rather high due to a large number of iterations. We attribute this to the dropping criterion. It can happen that in successive search sweeps connections from the same node are removed. This may accumulate to a rather large amount of dropped connections for such a node, if it is one of the last nodes to be eliminated. Due to this accumulation of local errors, the global error of the factorization can become too large, which results in a bad

problem size	drop tolerance	0	0.01	0.02	0.04	0.1	0.2	0.3	0.4	0.6	0.8
961	nnz per row	21.6	19.2	18.7	17.9	14.6	12.1	10.2	8.1	7.9	6.4
	iterations	1	2	2	3	4	5	7	11	12	19
3969	nnz per row	30.6	26.1	24.6	22.4	16.9	13.4	11.2	8.5	8.2	6.8
	iterations	1	2	2	3	4	6	10	14	16	30
16129	nnz per row	41.5	34.1	30.1	25.6	18.3	14.0	11.8	8.7	8.4	6.9
	iterations	1	2	2	3	5	9	14	19	26	49
65025	nnz per row	55.3	42.8	35.1	28.0	19.1	14.3	12.2	8.8	8.5	6.9
	iterations	1	2	3	3	5	11	18	29	38	97

Table 5.3: Results of OILU for approximate minimum degree ordering on Laplace equation as preconditioner in PCG

natural	drop tolerance	2e-4	4e-4	7e-4	1e-3	3e-3	5e-3	8e-3	2e-2	4e-2	1e-1
ordering	nnz per row	55.9	43.2	35.7	30.2	18.7	14.8	12.9	8.9	7.0	5.0
	iterations	5	7	8	10	19	24	28	41	55	97
amd	drop tolerance	1e-5	5e-5	1e-4	4e-4	2.5e-3	8e-3	1.5e-2	5e-2	1e-1	
ordering	nnz per row	45.4	38.8	35.4	28.1	18.9	14.6	11.7	9.0	7.0	
	iterations	2	3	4	8	19	31	44	73	105	

Table 5.4: Results of incomplete Choleski factorization in MATLAB with natural and approximate minimum degree (amd) ordering on Laplace equation as preconditioner in PCG. The problem size is 65025.

factorization. In MRILU [10] this is prevented by a sum criterion. We did not implement such a criterion which explains the deterioration of the results for large ε.

So far as solving is concerned this is done fastest by the direct solver. We don't need an iterative method to compute the solution and we can apply the factorization only once. This complies with the experiments in [9] where a similar observation was made in a comparison of methods to solve Poisson like equations. The bottleneck appears to be the construction phase and the amount of memory needed to store the factorization. From [10] we know that the amount of work for the construction goes down drastically when the fill decreases. For a similar case in [10] Table 1, the decrease of the relative fill from 8.7 to 3.6 brought the amount of work down by a factor 6. We expect a similar decrease here.

5.5.2 Convection-diffusion equation with grid stretching

To show that the idea can be as easily applied to a less trivial problem, we apply the method to a convection-diffusion equation on a unit square. The equation is given by

$$-\Delta u + u_x + 0.5u_y = 0 \text{ in } \Omega$$

problem	$\varepsilon = 0$		$\varepsilon = 0.01$		$\varepsilon = 0.1$		$\varepsilon = 0.2$		$\varepsilon = 0.3$	
size	r.fill	it	r.fill	it	r.fill	it	r.fill	it	r.fill	it
576	3.9	1	3.2	2	2.7	4	2.4	5	2.2	8
2401	5.3	1	4.3	3	3.6	5	2.9	8	2.6	12
9801	6.9	1	5.6	3	4.2	5	3.5	9	3.0	21
39601	8.7	1	7.6	3	5.1	6	4.0	13	3.3	40

Table 5.5: Results of OILU for approximate minimum degree ordering on convection-diffusion equation as preconditioner in PCG

with the same boundary conditions as used for the Poisson equation. In the discretization we use central differences. The Cartesian grid is refined regularly towards the left and lower boundary in such a way that the ratio of the largest and smallest mesh size is 100. In this case the iterations are started with the zero vector.

The results, given in Table 5.5, show that the method behaves very similar on this problem, however the number of iterations is slightly higher, which we attribute to the difficulty of convection and stretching.

5.5.3 Stokes

The third problem is the two-dimensional Stokes equation in a driven cavity. For that the following set of equations have to be solved on the unit square Ω

$$\left.\begin{array}{rcl} -\nu\Delta u + \nabla p &=& 0 \text{ in } \Omega, \\ \nabla \cdot u &=& 0 \text{ in } \Omega, \end{array}\right\} \tag{5.4}$$

where $u(x,y)$ is the velocity field and $p(x,y)$ the pressure field; the parameter ν controls the amount of viscosity. The non-trivial solution is determined by the boundary conditions that are zero on three sides of the unit square. At the upper boundary ($y = 1$) we have a horizontal velocity $u(x,1) = 1$.

We can get rid of the parameter ν by defining a new pressure variable $\bar{p} = p/\nu$. If the first equation is divided by ν, we can substitute p by $\bar{p}$ and the parameter ν is gone. So we may assume that $\nu = 1$.

The equations are discretized on a staggered grid (an Arakawa C-grid), which results in a system of linear equations $K\vec{u} = \vec{b}$, where K is of the form

$$\begin{bmatrix} A & B \\ B^T & 0 \end{bmatrix}.$$

Because this matrix is indefinite, we use clustering of variables, i.e. the velocity and pressure nodes of the same cell are treated as one variable. Thereby the 3×3 blocks on the diagonal are invertible. We compute an approximate minimum degree ordering for the block-matrix and

problem	$\varepsilon = 0$		$\varepsilon = 0.1$		$\varepsilon = 0.3$		$\varepsilon = 0.5$		$\varepsilon = 0.7$	
size	r.fill	it	r.fill	it	r.fill	it	r.fill	it	r.fill	it
147	4.1	1	4.0	2	3.7	3	3.4	6	3.3	7
675	6.4	1	6.3	3	5.9	5	5.2	9	4.8	12
2883	9.4	1	9.0	4	8.3	7	6.9	14	5.5	17
11907	13.7	1	13.1	4	11.0	8	8.2	21	6.1	45
48387	18.8	1	17.7	6	13.6	17	9.1	81	6.7	-

Table 5.6: Results for OILU for block approximate minimum degree ordering on Stokes equation as preconditioner in BICGSTAB.

apply the ordered incomplete LU factorization. The quality of the factorization is tested in BICGSTAB [81]. The results for several values of the drop tolerance can be found in Table 5.6.

Also here one observes that for large ε the increase of the fill with the problem size is less pronounced than for small ε. Unfortunately the number of iterations also increases rather fast for both increasing problem size and increasing ε. Even stagnation of convergence occurs for the largest problem for $\varepsilon = 0.7$. This is due to a too crude dropping criterion. Since the matrix is indefinite, the preconditioner might become close to singular. Clearly further research on robust dropping criteria is needed here.

Some readers may wonder why we are interested at all in preconditioners for K since there is a number of alternative approaches in which often only a system with A and one with a crude approximation, usual a diagonal matrix, to $B^T A^{-1} B$ needs to be solved. Since the system with A is of Poisson type, many methods are available to solve it, e.g. multi grid, MRILU, and also the current method (see Section 5.5.1). We will see some of these approaches in the next chapter. Our motivation to apply the ordered incomplete LU factorization is that by an accurate factorization of K we can use it as a preconditioner in eigenvalue problems in the way demonstrated in [73].

5.6 Conclusions

From the investigations in this chapter we may conclude the following.

1. The bare-branch ordering for a balanced elimination tree is useful for the construction of an ordered incomplete LU factorization. First of all it guarantees that the pruning by exact elimination is well balanced over the elimination tree. Furthermore it has a small number of levels even for large matrices. The inexact elimination can be placed naturally between the exact elimination of two consecutive levels. Any ordering of an elimination tree can be easily transformed to an equivalent bare-branch ordering.

2. The demand that no new fill is created in the Schur complement during the inexact elimination is sufficient to guarantee that the nodes in different subtrees do not get connected and that we do not ruin the fundamental fill-reducing ordering.

3. For the Poisson and the convection-diffusion problem we observe that the fill obtained by approximate minimum degree can be reduced significantly by inexact elimination, while the work in the solution phase increases only slightly (except when due to the too crude dropping criterion, too much is dropped in part of the matrix). From other sources [10] it is known that the reduction of the fill usually means a more than proportional reduction of the time needed for the construction of the factorization.

4. For the Stokes problem similar fill reductions can be obtained. However, the simple dropping criterion we used may lead to singularities in the preconditioner resulting in loss of speed of convergence. More sophisticated dropping criteria should remedy this situation.

At the end of this chapter we can conclude that the combination of incomplete LU factorizations and fill reducing orderings might be useful in case of Laplace type equations. However if this still holds for coupled partial differential equations is questionable. We made several attempts to improve the dropping criteria, but it hardly changed the results. Hence even if we find more sophisticated dropping criteria, it is unlikely the results in Table 5.6 will improve considerably. We didn't even mention the much more complex ocean system in equation (2.34). In both cases an approach that uses the structure of the equations (that is not seen due to clustering) might be better. We will study such methods in the next two chapters.

Chapter 6

Two preconditioners for saddle point problems in fluid flows

6.1 Introduction

Commonly used finite-element and finite-difference discretizations of the equations describing an incompressible flow, like for example the Navier-Stokes equations, lead to an equation $Kx = b$, where K is of saddle point type:

$$K = \begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix}. \tag{6.1}$$

In this matrix we have $A \in \mathbb{R}^{n\times n}$, a positive definite matrix, not necessarily symmetric, and $B \in \mathbb{R}^{n\times m}$, with $m \leq n$. Because of the zero diagonal block, K is indefinite, that is there are eigenvalues with positive real part as well as eigenvalues with negative real part.

In general there are two ways to solve large linear systems like the equation $Kx = b$: via a direct or an iterative method. If the problems become large direct methods require too much memory, so one is forced to use iterative methods. For an overview of commonly used Krylov subspace methods see [63, Chapter 6 and 7]. To speed up the convergence it is profitable to use a preconditioner. In this chapter we will focus on that for the case of saddle point matrices.

From literature a number of preconditioners for saddle point matrices is available. For a broad overview of the numerical solution of saddle point problems in all kind of applications see [6]. We will concentrate on incompressible flow problems. In Section 6.2 we recall the most important preconditioners for those kind of problems. In Section 6.3 we analyze two alternative preconditioners. In both sections we pay attention to the eigenvalues of the preconditioned saddle point matrix in order to get insight in the quality of the preconditioners and the convergence behavior in a Krylov subspace method. The results of the eigenvalue analysis is supported by numerical experiments presented in Section 6.4. We compare the preconditioners on three saddle point problems: the Stokes equation, the Oseen equation and an equation from ocean circulation.

6.2 Preconditioners from literature

The general idea of constructing a preconditioner for the saddle point matrix is to exploit the structure of the matrix in such a way that (reduced) systems result to which existing methods can be applied.

Given a preconditioner $\hat{K}$, the size and distribution of the eigenvalues of the generalized eigenvalue problem $Kx = \lambda \hat{K}x$ are important for the convergence of the Krylov subspace method. For many of the methods presented in this chapter the eigenvalues λ are related to the eigenvalues of the matrix $C = B^T A^{-1} B$, which is minus the Schur-complement of A in the saddle point matrix K. In general the matrix C will be dense, so one would never want to construct this matrix in practice. However in several cases, for example the Stokes equation, good estimates for the eigenvalues of C are available. In the remainder of the chapter we will assume that C is diagonizable and that (μ_i, q_i) is an eigenpair of C, so $Cq_i = \mu_i q_i$; there are m of these pairs.

6.2.1 SIMPLE and SIMPLER

The first two preconditioners we want to mention, are SIMPLE and SIMPLER. The original methods have been proposed by Patankar in [60] and are stand-alone iterative methods, based on a special splitting. We follow the formulation as a preconditioner of Vuik and Saghir [85] and use it in a Krylov subspace method. Both methods involve the matrix D, that is the diagonal of A, and an approximation to the Schur-complement $\hat{C}_{SI} = B^T D^{-1} B$. Because D is diagonal, $\hat{C}_{SI}$ will be sparse in contrast to C.

The SIMPLE preconditioner has the form:

$$\hat{K}_{SI} = \begin{pmatrix} A & AD^{-1}B \\ B^T & 0 \end{pmatrix} = \begin{pmatrix} A & 0 \\ B^T & I \end{pmatrix} \begin{pmatrix} I & D^{-1}B \\ 0 & -\hat{C}_{SI} \end{pmatrix}. \tag{6.2}$$

The factorization shows that solving an equation with $\hat{K}_{SI}$ can be reduced to solving equations with A and $\hat{C}_{SI}$.

Theorem 6.2.1. *The matrix $\hat{K}_{SI}^{-1}K$ has an eigenvalue 1 with multiplicity n. The remaining eigenvalues are equal to the eigenvalues of the matrix $\hat{C}_{SI}^{-1}C$.*

Proof. See [51]. We give a short proof here as well.

We can use the factorization of the SIMPLE preconditioner in Equation (6.2) to compute the matrix $\hat{K}_{SI}^{-1}K$:

$$\begin{pmatrix} I & D^{-1}B\hat{C}_{SI}^{-1} \\ 0 & -\hat{C}_{SI}^{-1} \end{pmatrix} \begin{pmatrix} A^{-1} & 0 \\ -B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} = \begin{pmatrix} I & A^{-1}B + D^{-1}B\hat{C}_{SI}^{-1}C \\ 0 & \hat{C}_{SI}^{-1}C \end{pmatrix}.$$

This matrix has an eigenvalue 1 with multiplicity n (the dimension of the heading block I) and the other eigenvalues are equal to the eigenvalues of the matrix $\hat{C}_{SI}^{-1}C$. □

Note that, even in the case of a symmetric K, the SIMPLE preconditioner is not symmetric, therefore its transpose $\hat{K}_{SI}^T$ could serve as a preconditioner as well. The transpose is called SIMPLEL (the "L" stems from "left"):

$$\hat{K}_{SL} = \begin{pmatrix} A & B \\ B^T D^{-1} A & 0 \end{pmatrix} = \begin{pmatrix} I & 0 \\ B^T D^{-1} & -\hat{C}_{SI} \end{pmatrix} \begin{pmatrix} A & B \\ 0 & I \end{pmatrix}.$$

The stand-alone iterative method SIMPLER combines both $\hat{K}_{SL}^{-1}$ and $\hat{K}_{SI}^{-1}$ in one iteration. If we formulate it as a preconditioner we get the following expression for the inverse of the SIMPLER preconditioner:

$$\hat{K}_{SR}^{-1} = \hat{K}_{SL}^{-1} + \hat{K}_{SI}^{-1} - \hat{K}_{SI}^{-1} K \hat{K}_{SL}^{-1} = \hat{K}_{SL}^{-1} + \hat{K}_{SI}^{-1} \left(I - K \hat{K}_{SL}^{-1} \right). \tag{6.3}$$

If K is symmetric, it holds that $\hat{K}_{SI} = \hat{K}_{SL}^T$ and consequently the operator $\hat{K}_{SR}^{-1}$ is symmetric. The last formulation in (6.3) offers the possibility to compute the action of the SIMPLER preconditioner on a vector efficiently in three consequent steps. The vector $x = \hat{K}_{SR}^{-1} y$ is computed via

$$\begin{cases} \text{solve } \hat{K}_{SL}\, \hat{x} = y, \\ \text{solve } \hat{K}_{SI}\, \tilde{x} = y - K\hat{x}, \\ \text{compute } x = \hat{x} + \tilde{x}. \end{cases} \tag{6.4}$$

For the eigenvalues of the SIMPLER preconditioned matrix we have the following result.

Theorem 6.2.2. *The matrix $\hat{K}_{SR}^{-1} K$ has an eigenvalue 1 with multiplicity $2m$.*

Proof. See [50]. □

The determination of the eigenvalues of $\hat{K}_{SR}^{-1} K$ is complicated, therefore we don't give a proof here. See [50] for the details. The remaining $n - m$ eigenvalues of the generalized eigenvalue problem are the eigenvalues of a matrix that is a function of A,D and B.

In general the SIMPLER preconditioner is more effective than SIMPLE and it is used quite often in industrial codes.

6.2.2 The preconditioner of Wathen and Silvester

In [71] the following block diagonal preconditioner is proposed,

$$\hat{K}_{WS}(\omega) = \begin{pmatrix} A & 0 \\ 0 & I/\omega \end{pmatrix}.$$

Note that a symmetric A gives a symmetric preconditioner. In the rest of the chapter we will call it the WS-preconditioner.
For the eigenvalues of the WS-preconditioned matrix we have the following theorem.

Theorem 6.2.3. *The matrix $\hat{K}_{WS}^{-1}(\omega) K$ has an eigenvalue 1 with multiplicity $(n-m)$. The remaining $2m$ eigenvalues are equal to $(1 \pm \sqrt{1 + 4\omega\mu_i})/2$.*

Proof. The proof can be found in [71], but we will give it here.

We solve the generalized eigenvalue problem $Kx = \lambda \hat{K}_{WS}(\omega)x$. First we split x in a natural way in two variables: $x^T = (u^T, p^T)$. Expanding the eigenvalue problem gives

$$(1-\lambda)Au + Bp = 0, \tag{6.5}$$
$$B^T u - \lambda p/\omega = 0. \tag{6.6}$$

Suppose $\lambda = 1$, then the term $(1-\lambda)Au$ drops out of the first equation. If we choose $p = 0$ and u in the kernel of B^T, the equations are satisfied. The dimension of $\ker(B^T)$ is $(n-m)$, so the multiplicity of the eigenvalue $\lambda = 1$ is $(n-m)$.

Now suppose that $\lambda \neq 1$. If we multiply Equation (6.5) from the left with BA^{-1} we get $(1-\lambda)B^T u + Cp = 0$. Equation (6.6) multiplied with $(1-\lambda)$ gives $(1-\lambda)B^T u - (1-\lambda)p/\omega = 0$. If we substract these two equations, we obtain

$$[C + (1-\lambda)\lambda/\omega I]\, p = 0,$$

which is just a reformulated eigenvalue problem for C. Now we replace p by q_i, one of the eigenvectors of C. Because $Cq_i = \mu_i q_i$, the equation will be satisfied if $[\mu_i + (1-\lambda)\lambda/\omega] = 0$. Solving λ from this expression gives

$$\lambda = -\frac{1 \pm \sqrt{1+4\omega\mu_i}}{2},$$

which proves the theorem. □

Usually $\omega = 1$ in case of the Stokes equation. In case of the Oseen equation $\omega = \nu$ (the viscosity), because the eigenvalues of C are $\mathcal{O}(1/\nu)$. However the method is not very sensitive to the precise value. If A is symmetric positive definite, C is also symmetric and positive definite, so the eigenvalues μ_i are real and we can assume that there is a largest and smallest eigenvalue: $0 \leq \mu_{\min} \leq \mu_i \leq \mu_{\max}$. If $\omega < 0$ all eigenvalues of the preconditioned matrix have positive real part and it becomes positive definite. For values $\omega < -1/(4\mu_{\max})$, the eigenvalues of the matrix $\hat{K}_{WS}^{-1}(\omega)K$ become complex. For large negative values of ω, the imaginary component becomes large as well and this will slow down the convergence.

If A is symmetric and positive definite, the eigenvalues μ_i of the Schur complement C are all positive and real. It then follows from the theorem that for positive values of ω the generalized eigenvalues are real and clustered in three regions. The first cluster coincides with the point 1, the second cluster exists of the negative eigenvalues $(1-\sqrt{1+4\omega\mu_i})/2$ and the third cluster contains the positive eigenvalues $(1+\sqrt{1+4\omega\mu_i})/2$. The clusters are clearly separated, therefore the convergence in a Krylov subspace method will be dominated by the two clusters away from 1.

6.2.3 The preconditioner of Elman and Silvester

In [26] Elman and Silvester studied the following non-symmetric preconditioner (from now on the ES-preconditioner)

$$\hat{K}_{ES}(\omega) = \begin{pmatrix} A & B \\ 0 & -I/\omega \end{pmatrix}.$$

Normally the preconditioner is used only if A is not symmetric. Nevertheless it can be used as a preconditioner for symmetric K as well and it even appears to be better than the WS-preconditioner. The advantage of the ES-preconditioner is that the preconditioned matrix has more eigenvalues equal to one and the distribution of the other eigenvalues is better as well. This is shown in the following theorem.

Theorem 6.2.4. *The matrix $\hat{K}_{ES}^{-1}(\omega)K$ has an eigenvalue 1 with multiplicity n. The remaining eigenvalues are equal to $\omega\mu_i$.*

Proof. The proof can be found in [26], but we will give it here as well.

We solve the generalized eigenvalue problem $Kx = \lambda\hat{K}_{ES}(\omega)x$. First we split x in a natural way in two variables: $x^T = (u^T, p^T)$. Expanding the eigenvalue problem gives

$$(1-\lambda)Au + (1-\lambda)Bp = 0, \tag{6.7}$$
$$B^T u + \lambda p/\omega = 0. \tag{6.8}$$

Suppose $\lambda = 1$, then the first equation becomes trivial. For each u we can construct $p = -\omega/\lambda B^T u$, to satisfy the second equation. For any u we have an eigenvector $(u^T, -\omega/\lambda u^T B)^T$ with eigenvalue 1. Since $u \in \mathbb{R}^n$, there are at most n independent vectors u. Therefore the multiplicity of the eigenvalue 1 is n.

Now suppose that $\lambda \neq 1$. If we multiply Equation (6.5) from the left with BA^{-1} and divide by $(1-\lambda)$ we get $B^T u + Cp = 0$. If we substract Equation (6.6) from this equation, we obtain

$$[C - \lambda/\omega I]\, p = 0.$$

Now we replace p by q_i, one of the eigenvectors of C. Because $Cq_i = \mu_i q_i$, the equation will be satisfied if $[\mu_i - (\lambda/\omega] = 0$. Solving λ from this expression gives

$$\lambda = \omega\mu_i,$$

which proves the theorem. □

Whereas $\hat{K}_{WS}(\omega)$ gives rise to three clusters, the theorem shows that here at most two clusters can occur. One cluster is around the eigenvalues $\omega\mu_i$ and if 1 is not in this cluster, there is another cluster namely the point 1 itself. The 'cluster' of all eigenvalues 1, can be approximated very fast by a Krylov subspace method, therefore the speed of convergence will be dominated by the scattering of the eigenvalues $\omega\mu_i$. In case of real symmetric A the eigenvalues are real and lie in the interval $[\mu_{max}, \mu_{min}]$. Convergence then is determined by the condition number $\mu_{\max}/\mu_{\min}$, which is equal to $\kappa(C)$. The parameter ω does not occur in the condition number, so its value appears to be arbitrary. We observed this ω-independency in numerical experiments that are not included in this chapter.

If A is nonsymmetric, the convergence is not related to the spectral condition number, but to the more general field of values. Nevertheless the convergence seems independent from the choice of the parameter ω.

6.2.4 The preconditioner of Kay, Loghin and Wathen

The fourth preconditioner we describe is different from the previous ones because it has been developed by Kay, Loghin and Wathen for a special class of saddle point problems, namely the Navier-Stokes equations. In the rest of the chapter we will refer to it as the KLW-preconditioner. We treat the preconditioner here because it is quite popular and as far as we know the best block-triangular preconditioner based on an approximation of the pressure Schur complement.

The incompressible Navier-Stokes equations on an open bounded domain are

$$\begin{aligned} -\nu\Delta u + (u\cdot\nabla)u + \nabla p &= 0, \\ \nabla\cdot u &= 0. \end{aligned} \tag{6.9}$$

The linearized version of this equation is

$$\begin{aligned} -\nu\Delta u + (w\cdot\nabla)u + \nabla p &= 0 \\ \nabla\cdot u &= 0 \end{aligned} \tag{6.10}$$

where the "wind" w is such that $\nabla\cdot w = 0$. These *Oseen equations* occur if we solve the Navier-Stokes equations via a Picard iteration, where we take $u = u^{(m)}$ and $w = u^{(m-1)}$, the previous guess for the solution.

After discretization of (6.10) we get a saddle point problem of the form (6.1) where A is of the form $\nu H + N$. Here H is the discrete Laplacian (Δ), a symmetric and positive definite matrix. The matrix N contains the convective part $(w\cdot\nabla)$ of the equations and is non-symmetric.

In [46] the following preconditioner is proposed

$$\hat{K}_{KLW} = \begin{pmatrix} A & B \\ 0 & -\hat{C}_{KLW} \end{pmatrix} \tag{6.11}$$

with special choice for $\hat{C}_{KLW}$

$$\hat{C}_{KLW} = B^T B A_p^{-1} M_p \tag{6.12}$$

where $A_p = \nu H_p + N_p$. Here the matrices H_p and N_p represent again the diffusive and convective part respectively, however they are now defined on the pressure space. The matrix M_p is the pressure-mass matrix. In our applications we use finite-difference or finite-volume methods and we scale the equations a priori, therefore we usually have $M_p = I$.

Note that $\hat{C}_{KLW} = C$ would imply that all eigenvalues of the preconditioned matrix $\hat{K}_{KLW}^{-1}K$ are one. In that case m of these eigenvalues are defective. Defective eigenvalues with large geometric multiplicity can slow down the convergence in the Krylov subspace method. Fortunately we can show that $(\hat{K}_{KLW}^{-1}K - I)^2 = 0$, so the geometric multiplicity of all eigenvalues is at most 2.

In general the KLW-preconditioner will perform well if $\hat{C}_{KLW}$ as defined in Equation (6.12) is a good approximation to the true Schur complement. There are several ways to derive the given expression. In [46] it is done via Green's functions. We will use an argument that is given in [27].

The first step is the search for a matrix A_p such that

$$AB \approx BA_p. \tag{6.13}$$

We want this approximation to be as good as possible. This appears to be the case if A_p is a problem similar to A, but defined on the pressure space. An heuristic argument is the fact that in an infinite dimensional space the gradient and Laplace operator commute. Left-multiplication with $B^T A^{-1}$ gives $B^T B \approx C A_p$. A rearrangement of the equation results in the following approximation to the Schur complement:

$$B^T B A_p^{-1} \approx C. \tag{6.14}$$

Note that $B^T B$ is the discrete form of $\nabla \cdot \nabla = \Delta$ on the pressure space, therefore $B^T B = H_p$. In case of the Stokes equation (where the wind field $w = 0$, and so $N_p = 0$) it is nice if the preconditioner coincides with the ES-preconditioner (as described in Section 6.2.3). This is ensured by an additional scaling of the approximation with the pressure-mass matrix, which gives us (6.12).
Application of the KLW-preconditioner requires the solution of an equation with $\hat{C}_{KLW}$. Solving that equation requires the application of A_p to a vector and the solution of an equation with $B^T B$, which is quite easy because it represents a discretization of a Laplacian.

The preconditioner introduces the matrix A_p, which is related to an artificial convection-diffusion equation in the pressure space. We have to determine appropriate boundary conditions for this problem and its matrix. The derivation and motivation of the preconditioner do not give a clear indication what conditions we should impose. In [46] and [27] it is advised to chose standard Neumann boundary conditions in case of velocity boundary conditions. The main argument is that the pressure field is determined up to a constant and that Neumann boundary conditions keep this property for A_p.

For the eigenvalues of the KLW-preconditioned matrix we have the following result.

Theorem 6.2.5. *The matrix $\hat{K}_{KLW}^{-1} K$ has an eigenvalue 1 with multiplicity n. The remaining eigenvalues are equal to the eigenvalues of the matrix $\hat{C}_{KLW}^{-1} C$.*

Proof. The eigenvalues of $\hat{K}_{KLW}^{-1} K$ are equal to the ones of $K \hat{K}_{KLW}^{-1}$. The last matrix is equal to

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} A^{-1} & A^{-1} B \hat{C}_{KLW}^{-1} \\ 0 & -\hat{C}_{KLW}^{-1} \end{pmatrix} = \begin{pmatrix} I & 0 \\ B^T A^{-1} & -C \hat{C}_{KLW}^{-1} \end{pmatrix}.$$

From this matrix we get the eigenvalues 1 (n times) and the eigenvalues of $C \hat{C}_{KLW}^{-1}$ which are the same as the ones of $\hat{C}_{KLW}^{-1} C$. □

Now we have the same problem as with the eigenvalues of the SIMPLE preconditioned matrix (note the similarity between this theorem and Theorem 6.2.1). The eigenvalues of $\hat{C}_{KLW}^{-1} C$ are hard to determine. There are merely numerical results that show that most eigenvalues in case of the Oseen equations are clustered around 1. Only a few eigenvalues can be found outside a small circle around 1. Theoretical results in the form of bounds on the eigenvalues are only obtained for the more "symmetric" approximation to the Schur complement $M_p^{1/2} (B^T B)^{1/2} A_p^{-1} (B^T B)^{1/2} M_p^{1/2}$, that is never used in practice. See [27] for these results.

6.3 Two alternative preconditioners

We want to discuss two alternative preconditioners for saddle point equations. Both are symmetric if K is symmetric.

6.3.1 A preconditioner based on the augmented Lagrangian

The first preconditioner can be viewed as a variant of the WS-preconditioner that we described in Section 6.2.2. The difference: we add a term ωBB^T to the matrix A. This results in what we will call the grad-div preconditioner

$$\hat{K}_{GD}(\omega) = \begin{pmatrix} A+\omega BB^T & 0 \\ 0 & I/\omega \end{pmatrix}.$$

If A is symmetric positive definite and $\omega > 0$, we have a symmetric and positive definite preconditioner $\hat{K}_{GD}$ for the symmetric and indefinite matrix K.

Let us define the following transformation matrix

$$T(\omega) = \begin{pmatrix} I & \omega B \\ 0 & I \end{pmatrix}. \tag{6.15}$$

Application of the transformation with $\omega > 0$ to a saddle point equation $Kx = b$ where K has the structure (6.1) gives

$$T(\omega)Kx = T(\omega)b, \tag{6.16}$$

where

$$T(\omega)K = \begin{pmatrix} A+\omega BB^T & B \\ B^T & 0 \end{pmatrix}. \tag{6.17}$$

Note that $T(\omega)K$ inherits symmetry from K.
Equation (6.16) is well known as the augmented Lagrangian formulation of the saddle point problem [35, 7]. In case of incompressible flow problems it is also called grad-div stabilization, because B and B^T are the discrete versions of the gradient and the divergence operator respectively and the addition of the term ωBB^T allows a more accurate solution of the velocity field for small viscosity [57, 58].

The most important difference between the GD-preconditioner and the augmented Lagrangian approach is that we use the augmented preconditioner for the non-augmented matrix (6.1). The preconditioner is in fact the WS-preconditioner for $T(\omega)K$ used as preconditioner for K itself. It seems that we use the wrong preconditioner for K, which makes this approach a little strange, but the eigenvalue analysis will show the advantages.

We have the following theorem about the eigenvalues of the grad-div preconditioned matrix.

Theorem 6.3.1. *The matrix $\hat{K}_{GD}^{-1}(\omega)K$ has an eigenvalue 1 with multiplicity n. The remaining eigenvalues are equal to*

$$-\frac{\omega\mu_i}{1+\omega\mu_i}.$$

Proof. We solve the generalized eigenvalue problem $Kx = \lambda\hat{K}_{GD}(\omega)x$. First we split x in a natural way in two variables: $x^T = (u^T, p^T)$. Expanding the eigenvalue problem gives

$$\begin{aligned}(1-\lambda)Au - \lambda\omega BB^T u + Bp &= 0, &(6.18)\\ B^T u - \lambda p/\omega &= 0. &(6.19)\end{aligned}$$

Suppose $\lambda = 1$, then the term $(1-\lambda)Au$ drops out of the first equation. The remainder can be written as $-\omega B(B^T u - p/\omega) = 0$. Between the brackets we recognize the second equation, so if that equation is satisfied, the first becomes trivial. Because the first equation reduces to a trivial one, we can choose u totally free. For any u we have an eigenvector $(u^T, \omega u^T B)^T$ with eigenvalue 1. Since $u \in \mathbb{R}^n$, there are at most n independent vectors u. Therefore the multiplicity of the eigenvalue 1 is n.

Now suppose that $\lambda \neq 1$. We rewrite Equation (6.19) to $B^T u = \lambda p/\omega$ and substitute $B^T u$ in Equation (6.18). We end up with

$$(1-\lambda)Au + (1-\lambda^2)Bp = 0.$$

We have chosen $\lambda \neq 1$ and A is non-singular, so we can invert $(1-\lambda)A$ and transform this equation into $u = -(1+\lambda)A^{-1}Bp$. This u can be substituted in Equation (6.19), which gives

$$-\left[(1+\lambda)C + \lambda/\omega I\right] p = 0,$$

which is just a reformulated eigenvalue problem for C. Now we replace p by q_i, one of the eigenvectors of C. Because $Cq_i = \mu_i q_i$, the equation will be satisfied if $[(1+\lambda)\mu_i + \lambda/\omega] = 0$. Solving λ from this expression gives

$$\lambda = -\frac{\omega\mu_i}{1+\omega\mu_i},$$

which proves the theorem. □

For positive ω and μ_i real, the eigenvalues are contained in $(-1,0)\cup\{1\}$. Note that we cannot choose ω equal or close to $-1/\mu_i$, because then $1+\omega\mu_i$ will become very small and we get a very big eigenvalue. We can choose a negative value $\omega < -1/\mu_{\min}$. However we need an estimate of the value of $\mu_{\min}$ to choose an appropriate ω. In general we do not have that, therefore we choose positive ω.

It is remarkable that for $\omega \to \pm\infty$ we have only two eigenvalues namely ± 1. Therefore, for large values of $|\omega|$ the Krylov subspace method should converge in few iterations. However in practice it is not possible to use large ω, because then the grad-div added system will become almost singular.

One more remark on the eigenvalues. If we compare the grad-div preconditioner with the preconditioner of Wathen and Silvester we see that by simply adding ωBB^T to A we half the number of eigenvalues that is not equal to one and the remaining eigenvalues are scaled and shifted such that they are much closer to -1.

6.3.2 A preconditioner based on artificial compressibility

The second preconditioner that we analyze in this section, is constructed by adding artificial compressibility to the matrix K. This means that the zero diagonal block is replaced by the identity matrix multiplied with the parameter $-1/\omega$:

$$\hat{K}_{AC}(\omega) = \begin{pmatrix} A & B \\ B^T & -I/\omega \end{pmatrix}. \tag{6.20}$$

The preconditioner can be viewed as a variant of the ES-preconditioner (see Section 6.2.3). We obtain $\hat{K}_{AC}(\omega)$ by replacing the zero block in $\hat{K}_{ES}$ by B^T.

The approach is not completely new. Almost three decades ago in [5] it was proposed as preconditioning by regularization. Maybe due to the lack of support by numerical results in that paper it has not been used much. In [54] we studied the spectral properties of the preconditioner. Recently a similar approach was used in [20, 30] where it is called a primal-based penalty preconditioner. In the last paper a preconditioner is proposed for more general saddle point problems where the lower (2,2) block in (6.1) can be any negative semidefinite matrix. The AC-preconditioner we describe here can be seen as a special case of the primal-based penalty preconditioner. The preconditioner in [20] is applied to incompressible elasticity problems. We will apply our preconditioner to incompressible flow equations as is done in [30].

In analogy to the case of the grad-div preconditioner, the eigenvalues of the matrix K, preconditioned with $\hat{K}_{AC}(\omega)$, are related to the eigenvalues of the Schur complement. This is stated in the following theorem.

Theorem 6.3.2. *The matrix $\hat{K}_{AC}^{-1}(\omega)K$ has an eigenvalue 1 with multiplicity n. The remaining eigenvalues are equal to*

$$\frac{\omega\mu_i}{1+\omega\mu_i}.$$

Proof. First we expand the generalized eigenvalue problem $Kx = \lambda\hat{K}_{AC}(\omega)x$. with $x^T = (u^T, p^T)$.

$$\begin{aligned} (1-\lambda)Au + (1-\lambda)Bp &= 0, \\ (1-\lambda)B^T u + \frac{\lambda}{\omega}p &= 0. \end{aligned}$$

If $\lambda = 1$, these equations reduce to $p/\omega = 0$. So p has to be zero and u is totally free. Any vector $(u^T, 0)^T$ is an eigenvector with eigenvalue 1. Since $u \in \mathbb{R}^n$, the multiplicity of eigenvalue 1 is n.

Now consider the case where $\lambda \neq 1$. We premultiply the first equation with $B^T A^{-1}$ in order to get $(1-\lambda)B^T u + (1-\lambda)Cp = 0$. If we subtract this equation from the second one we get

$$\frac{\lambda}{\omega}p - (1-\lambda)Cp = 0.$$

In this equation we replace p by q_i, one of the eigenvectors of C. Now we use $Cq_i = \mu_i q_i$ and we have $(\lambda/\omega - \mu_i + \lambda\mu_i)q_i = 0$. The expression between brackets becomes zero if

$$\lambda = \frac{\omega\mu_i}{1+\omega\mu_i},$$

which proves the theorem. □

For μ_i real and positive values of ω, the eigenvalues are contained in the semi open interval $(0,1]$. For negative values of ω we have the same conditions as for $\hat{K}_{WS}$: ω should not be equal to $-1/\mu_i$ and if $\omega < -1/\mu_{\min}$ the eigenvalues are in $[1,\infty)$.

If we send ω to $\pm\infty$ all eigenvalues of the preconditioned matrix converge to 1. Here it is less surprising then in case of the grad-div preconditioner, because the matrix $\hat{K}_{AC}(\omega)$ converges to K for large ω.

The eigenvalues of the matrices $\hat{K}_{GD}^{-1}(\omega)K$ and $\hat{K}_{AC}^{-1}(\omega)K$ are the same, except for a minus sign in front of the eigenvalues that are not one. This suggests some relation between the two preconditioners. Indeed there is one and it can be easily understood from the following factorization

$$\begin{pmatrix} A & B \\ B^T & -I/\omega \end{pmatrix} = \begin{pmatrix} I & -\omega B \\ 0 & I \end{pmatrix} \begin{pmatrix} A+\omega BB^T & 0 \\ 0 & -I/\omega \end{pmatrix} \begin{pmatrix} I & 0 \\ -\omega B^T & I \end{pmatrix}. \tag{6.21}$$

This shows that $\hat{K}_{AC}(\omega)$ is a composition of transformations $T(-\omega)$, $T(-\omega)^T$, defined in Equation (6.15), and a block diagonal matrix that is almost equal to $\hat{K}_{GD}$ except for a minus sign in the second diagonal block.

The factorization is useful not only for theoretical purposes. It provides a way to compute the action of $\hat{K}_{AC}^{-1}(\omega)$. The inverses of the transformation matrices are easy ($T(-\omega)^{-1} = T(\omega)$), so the remaining problem is to solve an equation with $A+\omega BB^T$ in the block diagonal matrix.

One more remark on the eigenvalues of the preconditioned matrix. In [7] the authors use an ES-preconditioner (with parameter $-(\omega+\nu)$) for the augmented system $T(\omega)K$. The generalized eigenvalues then become $(\omega+\nu)\mu_i/(1+\omega\mu_i)$. Note that these are almost the same as the ones we obtained in Theorem 6.3.2 for the AC-preconditioner. We can explain the similarity by transformations with $T(\omega)$. If we ignore the parameter ν (which is very small compared to ω in most cases), the ES-preconditioner in [7] is

$$\begin{pmatrix} A+\omega BB^T & B \\ 0 & -I/\omega \end{pmatrix} = \begin{pmatrix} A & B \\ B^T & -I/\omega \end{pmatrix} \begin{pmatrix} I & 0 \\ \omega B^T & I \end{pmatrix} = \hat{K}_{AC}(\omega)T(\omega)^T. \tag{6.22}$$

This preconditioner is used for the matrix $T(\omega)K$ which is equal to $KT(\omega)^T$. Note that both the preconditioner and the matrix have the same factor. In the generalized eigenvalue problem it would drop out and we get the very same eigenvalue problem as we had for the AC-preconditioner. If we forget the ν and consider right-preconditioning, the preconditioner in [7]

applied to the augmented system is equivalent to an AC-preconditioner applied to the original saddle point problem. This explains why the results we obtain in Section 6.4.2 for the Oseen equations are similar to the results in [7].

Instead of (6.22) we can consider the ES-preconditioner with $+\omega$ as parameter as a preconditioner for the augmented system. We have a similar factorization:

$$\begin{pmatrix} A+\omega BB^T & B \\ 0 & I/\omega \end{pmatrix} = \begin{pmatrix} I & \omega B \\ 0 & I \end{pmatrix} \begin{pmatrix} A+\omega BB^T & 0 \\ 0 & I/\omega \end{pmatrix} = T(\omega)\hat{K}_{GD}(\omega).$$

Now this preconditioner and the augmented system have a factor $T(\omega)$ in common. Using this preconditioner as a left-preconditioner for the augmented system is equivalent to solving the non-augmented system with a GD-preconditioner.

6.3.3 Remarks on the condition number

If one of the preconditioners $\hat{K}_*$ in this chapter is used in a Krylov subspace method, the convergence will depend on the distribution of the eigenvalues of the preconditioned matrix over the complex plane. In case of symmetric positive definite A the distribution can be measured by the spectral condition number κ, which is the quotient of the largest and smallest eigenvalue in absolute value.

In case of the preconditioners in this chapter the condition number appears not to be a good estimate. This is caused by the large number of eigenvalues 1. In itself it is already advantageous when eigenvalues coincide, since the number of iterations of a Krylov subspace method is at most equal to the number of different eigenvalues (assuming they are simple). However, if these coinciding eigenvalues are also well separated from the others they become extreme eigenvalues, of which the corresponding subspace is found first by the subspace method, and after that these eigenvalues do not play a role anymore in the speed of convergence, see [82, §5.3, §6.2]. Therefore we will use an effective spectral condition number $\tilde{\kappa}$, instead of the real spectral condition number κ. The number $\tilde{\kappa}$ is the quotient of the largest and smallest eigenvalue in absolute value *not equal to one*.

For the ES-preconditioner we have

$$\tilde{\kappa}(\hat{K}_{ES}^{-1}(\omega)K) = \frac{\omega\mu_{\max}}{\omega\mu_{\min}} = \frac{\mu_{\max}}{\mu_{\min}} = \kappa(C). \tag{6.23}$$

The nonunit eigenvalues of the preconditioned matrices $\hat{K}_{AC}^{-1}K$ and $\hat{K}_{GD}^{-1}K$ are the same except for a minus sign. Hence, their condition numbers are equal. Since, the function $x/(1+x)$ is monotonously increasing, the largest eigenvalue of the preconditioned matrix corresponds to the largest eigenvalue $\mu_{\max}$ and the smallest to the smallest eigenvalue $\mu_{\min}$. The effective condition number of both methods is

$$\tilde{\kappa}(\hat{K}_{AC}^{-1}K) = \tilde{\kappa}(\hat{K}_{GD}^{-1}K) = \frac{\omega\mu_{\max}}{1+\omega\mu_{\max}} \cdot \frac{1+\omega\mu_{\min}}{\omega\mu_{\min}} = \frac{\mu_{\max}}{\mu_{\min}} \cdot \frac{1+\omega\mu_{\min}}{1+\omega\mu_{\max}} \leq \kappa(C). \tag{6.24}$$

The eigenvalue analysis already showed that the preconditioners are good for large ω. This analysis shows that for any positive value of ω, the effective condition number of the preconditioners is better than that of the ES-preconditioner at least in the case of symmetric and positive definite A.

6.3.4 Solving grad-div added systems

An important issue for the AC- and GD-preconditioner is the solution of grad-div added systems. The alternative preconditioners have better spectral properties than the ES- and WS-preconditioner, but they require the solution of systems involving $A+\omega BB^T$. This system is more difficult to solve than A itself. If we solve this system using a (preconditioned) Krylov subspace method, we get so called *nested iterations*. For each iteration step in the outer Krylov method, we perform a number of inner iterations to solve A or $A+\omega BB^T$. If we apply one of the preconditioners that we introduced in this section, the gain in the outer iterations could get lost in the solution of systems in the inner iterations.

The solution of systems with A is not an issue at all. In the applications in this chapter the matrix A is of convection-diffusion type with possibly a Coriolis force. For these types of systems many (algebraic) multigrid methods and incomplete LU factorizations are available that provide good preconditioners [10, 28]. With the addition of ωBB^T to A it becomes questionable whether these methods still can be applied. Therefore in this section we will discuss ways to solve the grad-div added systems.

In general the addition of ωBB^T to A will create new entries in the matrix. The increase of non zeros depends on the underlying equations and their discretization. For the problems we discuss in the next section the number of non zeros is almost doubled by grad-div adding.

A second remark: if the matrix A is an M-matrix, which means it is symmetric positive definite and diagonally dominant, the matrix $A+\omega BB^T$ will still be symmetric and positive definite (as long as $\omega > 0$), but it will loose diagonal dominance; there will be positive off-diagonal entries. For M-matrices a lot of methods are available. In general they cannot be applied without any concern to the matrix $A+\omega BB^T$. However in case of the test problems we describe in the next section, we will see that for moderate values of ω some of these methods can be applied.

Furthermore we want to point out that the matrices A and $A+\omega BB^T$ are spectrally equivalent. The matrix $A^{-1}(A+\omega BB^T)$ has $n-m$ eigenvalues 1 and the other m eigenvalues are $1+\omega\mu_i$. The effective condition number is

$$\tilde{\kappa}(A^{-1}(A+\omega BB^T)) = \frac{1+\omega\mu_{\max}}{1+\omega\mu_{\min}} \leq \kappa(C).$$

If this last number is small, the matrix A can be used as preconditioner for $A+\omega BB^T$. Instead of the matrix A itself we can also use a preconditioner for A as a preconditioner for $A+\omega BB^T$.

Finally we have to remark that the theory about solution of grad-div added systems is not well developed yet. Several papers struggle with the question what is the best way to solve this kind of systems. [58] uses a standard multigrid method, [20, 30] a balancing domain

decomposition by constraints (BDDC) preconditioner and [7] a multigrid method with a special type of Gauss-Seidel smoother. The last multigrid method seems the most promising for our applications. At least for the Oseen equations the convergence is independent of the mesh size. For all preconditioners it holds that the results deteriorate with increasing ω.

Summarizing we can state that multigrid methods exist for the solution of grad-div added systems that work fine for modest values of ω. Unfortunately we do not have these codes, so in the next section we will use a robust standard ILU-factorization for the grad-div added systems. Maybe an ILU-factorization is not the best method, but for modest values it suffices to illustrate the possibilities of the preconditioners.

The research for a better and more general approach for grad-div added systems especially for larger values of ω is ongoing.

6.4 Numerical results

In this section we show a comparison between the preconditioners for two different saddle point problems. All computations are performed in MATLAB (7.1.0.183 (R14) Service Pack 3) on a PC with two 2.4GHz AMD Opteron processors and 7.6GB memory.

6.4.1 Stokes flow in a driven cavity

The first problem is the two-dimensional Stokes equation in a driven cavity. The following set of equations has to be solved on the unit square Ω

$$\left.\begin{array}{rcl} -\nu\Delta u + \nabla p &=& 0, \\ \nabla \cdot u &=& 0. \end{array}\right\} \tag{6.25}$$

where $u(x,y)$ is the velocity field and $p(x,y)$ the pressure field; the parameter ν controls the amount of viscosity. The non-trivial solution is determined by the boundary conditions that are zero on three sides of the unit square. At the upper boundary ($y = 1$) we have a horizontal velocity $u(x,1) = 1$.

We can get rid of the parameter ν by defining a new pressure variable $\bar{p} = p/\nu$. If the first equation is divided by ν, we can substitute p by $\bar{p}$ and the parameter ν is gone. So we may assume that $\nu = 1$.

The equations are discretized on a uniform staggered grid (a Marker-and-Cell or C-grid) with mesh size h using standard second-order finite differences, which results in a system of linear equations $Kx = b$, where K is of saddle point form. It is well known that in this case the Schur complement $C = BA^{-1}B^T \sim I$, hence the condition number $\kappa(C)$ is independent of the mesh size.

In Table 6.1 we show the number of iterations in BICGSTAB [81] needed to obtain an accuracy of 10^{-6} for the preconditioned Stokes matrix. We applied the preconditioners mentioned in this chapter (except KLW, because it coincides with ES) and varied the parameter ω in case

METHOD	SI	SR	WS	ES	GD			AC		
ω	-	-	1	1	1	16	256	1	16	256
h = 1/32	48	8	15	7	5	3	3	4	2	2
1/64	111	12	18	7	5	3	3	4	2	2
1/128	243	14	20	7	5	3	2	4	2	2
1/256	>300	22	23	7	5	3	2	4	2	2

Table 6.1: Number of iterations in BICGSTAB (accuracy 10^{-6}) with several preconditioners for the Stokes equation in a driven cavity. Subsystems are solved exactly.

of the artificial compressibility and grad-div preconditioner. Moreover we used several mesh sizes. Equations with A or $A+\omega BB^T$ are solved via an exact LU factorization.

The results are as we expected from the eigenvalue analysis. The condition number $\kappa(C)$ is independent of the mesh size and so is the number of iterations using the ES-, GD- and AC-preconditioners. For the SIMPLER and WS-preconditioner the number of iterations slightly increases when the mesh size decreases, so there is some grid dependence. The SIMPLE preconditioner cannot be considered as a serious alternative, because the number of iterations grows rapidly with the grid size.

As one can see, even for ω equal to one, the number of iterations for the GD- and AC-preconditioner is smaller than for the ES-preconditioner. According to (6.24) the effective condition numbers of the GD- and AC-preconditioned matrices are smaller than that of the ES-preconditioned matrix. The last number is equal to the condition number of C, that is independent of the grid size. Indeed BICGSTAB converges almost two times faster for the GD- and AC-preconditioner. With increasing ω, the effective condition numbers of GD and AC get even closer to 1 and the number of iterations decreases further, as we expected from the analysis.

We have to remark that in Table 6.1 we only measured the number of iterations. The amount of work per iteration per preconditioner varies quite a lot. For example one iteration with SIMPLER requires a lot more computations than one iteration with the AC-preconditioner, which in turn is more expensive than one iteration with the ES-preconditioner. Therefore in Table 6.2 we attempt to make a fairer comparison between the methods. We compare the CPU-times for the construction of the preconditioner and the solution of the largest Stokes problem. We do not use exact solves for the subsystems A and $A+\omega BB^T$, but instead we solve these system with GMRES using MATLAB's LUINC factorization as a preconditioner. The drop tolerance for the incomplete factorization is $3\cdot 10^{-4}$ in case of A and 10^{-4} in case of $A+\omega BB^T$. The tolerances are chosen such that both factorizations have approximately the same amount of nonzeros per row. Because of the nested iterative scheme we have to use a flexible Krylov method in the outer iteration. In the experiments we used FGMRES [62]. Because we use a flexible Krylov method, there is no need to solve the inner iterations very accurately. We stop the inner iteration as soon as an accuracy of 10^{-3} is reached. The accuracy for the outer iteration is 10^{-6}.

The table shows that the grad-div and artificial compressibility preconditioners can compete

METHOD	SR	WS	ES	GD	AC
VALUE OF ω	-	1	1	16	16
ITERATIONS IN FGMRES	30	26	13	5	4
CONSTRUCTION TIME (SEC)	18	11	11	22	22
SOLVE TIME (SEC)	1822	456	214	171	113

Table 6.2: Number of iterations in FGMRES and CPU-times required for the solution of the Stokes equation with mesh size $h = 1/256$. Subsystems are solved with one application of MATLABS's LUINC with drop tolerance $3 \cdot 10^{-4}$ (WS/ES) or 10^{-4} (GD/AC).

in practice with the ES- and WS-preconditioners. The construction of the first preconditioners is more expensive because we have to compute an incomplete LU factorization for the grad-div added matrix that has more nonzeros than the original one. Note that even with the relative large value $\omega = 16$ we are able to compute a good incomplete factorization for the matrix $A + \omega BB^T$. So the problems sketched in Section 6.3.4 appear to be not too serious in case of the Stokes equation. The loss in time in the factorization phase of the GD- and AC-preconditioner is compensated by a much faster convergence of the outer iteration in the solution phase.

Finally we remark that the table illustrates that SIMPLER is much more expensive to apply then the other methods. It requires approximately the same amount of iterations as the WS-preconditioner, but it is per iteration roughly four times more expensive.

6.4.2 Oseen equations in a driven cavity

The second test is a benchmark problem that is proposed in [27] and used in [7]. It is again a driven cavity problem, but now for the Oseen equations that we described in Equation (6.10). The area Ω is the unit square. The boundary conditions are

$$\begin{cases} u_1 = u_2 = 0 & \text{for} \quad x = 0, x = 1 \text{ and } y = 0 \\ u_1 = 1, u_2 = 0 & \text{for} \quad y = 1 \end{cases} \tag{6.26}$$

the wind is

$$w = \begin{pmatrix} 2(2y-1)(1-(2x-1)^2) \\ -2(2x-1)(1-(2y-1)^2) \end{pmatrix}. \tag{6.27}$$

This wind contains a single recirculation on the domain Ω. For a more detailed description of the problem see [27]. We solve the problem on a regular grid with a Marker-and-Cell or C-grid discretization using central differences.

We do not show the results of the SI-, SR-, WS- and ES-preconditioner because they perform quite bad on the Oseen equations. The best preconditioner in this case is the KLW-preconditioner so we will compare that one to the artificial compressibility preconditioner. We do not give the results for the GD-preconditioner, as they are similar to the ones of the AC-preconditioner. Sometimes the GD-preconditioner requires a few more iterations, like in the results for the Stokes equation.

	VISCOSITY ν				
	1/20	1/40	1/80	1/160	1/320
h = 1/16	17	19	21	24	26
1/32	17	19	21	22	24
1/64	18	20	21	23	25
1/128	19	20	22	23	25
1/256	18	19	22	23	25

Table 6.3: Number of iterations in GMRES (accuracy 10^{-6}) for the KLW-preconditioner for the Oseen equations in a driven cavity, subsystems are solved exactly.

	VISCOSITY ν				
	1/20	1/40	1/80	1/160	1/320
h = 1/16	6	6	6	6	6
1/32	6	6	6	6	6
1/64	5	5	5	6	6
1/128	5	5	5	5	5
1/256	4	4	4	5	5

Table 6.4: Number of iterations in GMRES (accuracy 10^{-6}) for the artificial compressibility preconditioner with $\omega = 1$ for the Oseen equations in a driven cavity, subsystems are solved exactly

The results with the KLW-preconditioner for our discretization can be found in Table 6.3. The number of iterations is independent of the mesh size, but slightly depends on the viscosity. This agrees with the results found in [27, 7].

In Table 6.4 one finds the results of preconditioning with the artificial compressibility preconditioner for the Oseen equations. For the parameter ω we chose the value 1. Clearly the number of iterations is independent of grid size and viscosity for this value of ω.

The results we obtain are comparable to the ones in [30] on a slightly different Oseen problem. In that paper the number of outer iteration seems to increase a little with the Reynolds number. We do not observe this in Table 6.4. This difference could be explained by the choice of the parameter ω.

We show the dependency on ω in Table 6.5. The table contains the number of iterations for different values of ω and ν. Only for the smallest value $\omega = 1/16$ the number of iterations depends slightly on ν. For even smaller values of ω (for example $1/64$) the number of iterations increases more rapidly with decreasing viscosity and eventually the method will fail to converge. This is not a surprise, because for these values the preconditioner becomes more and more similar to the ES-preconditioner which is known to be not appropriate for the Oseen equation with small viscosity.

The number of iterations does not tell the whole story. In both Tables 6.3 and 6.4 we solved the subsystems with A and $\hat{C}_{KLW}$ or $A + \omega BB^T$ exactly. In general the last system will be more difficult to solve. Therefore in Table 6.6 we compare the CPU-times for the largest problem

	VISCOSITY ν				
	1/20	1/40	1/80	1/160	1/320
$\omega = 4$	3	3	3	3	3
1	4	4	4	5	5
1/4	8	8	9	9	9
1/16	14	17	21	23	25

Table 6.5: Number of iterations in GMRES (accuracy 10^{-6}) for the artificial compressibility preconditioner with several values of ω for the Oseen equations in a driven cavity on the largest grid $h = 1/256$, subsystems are solved exactly

METHOD	KLW	KLW	AC	AC
VISCOSITY ν	1/20	1/320	1/20	1/320
ITERATIONS IN FGMRES	26	48	13	11
CONSTRUCTION TIME (SEC)	11	11	18	12
SOLVE TIME (SEC)	148	277	66	147

Table 6.6: Number of iterations in FGMRES (accuracy 10^{-6}) and CPU-times required to solve the Oseen equations for $h = 1/256$. In all cases $\omega = 1$. Subsystems are solved with GMRES up to an accuracy of 10^{-3} using MATLAB's LUINC factorization with drop tolerance $1 \cdot 10^{-4}$ (KLW) and $3 \cdot 10^{-4}$ (AC).

($h = 1/256$) and for the two extremal values of ν. When we have to apply the preconditioner we solve the subsystems with GMRES using the incomplete LU factorization of MATLAB as preconditioner, which is certainly not the best for these problems, but here it suffices. We chose the drop tolerance such that all incomplete factorizations have approximately the same amount of fill per row.

In Table 6.7 we show the dependency of the CPU-times on the grid size. The grid is refined with a factor 2 in each direction. For the larger problems the solve time grows with a factor bigger than 4. Nor KLW nor AC combined with LUINC is ideal in the sense that the amount of work scales linearly with the problem size. In the table we see that the increase in time is caused mainly by the number of inner iterations. This is observed in [30] as well. Here we can blame LUINC. We already mentioned that there exist better (multigrid) methods to solve A (see [46]) and $A + \omega BB^T$ (see [7]), but, although the LUINC-factorization is not the best, the scaling results are reasonable.

The timing results show that in case of the Oseen equation the artificial compressibility preconditioner is not only theoretically but also in practice a serious alternative to the KLW-preconditioner.

6.4.3 Thermohaline ocean circulation

One of the applications of the sketched preconditioners is the numerical simulation of three dimensional thermohaline ocean circulation. The equations of thermohaline ocean circulation

METHOD	KLW				AC			
	OUTER ITER	INNER ITER	CONSTR. TIME	SOLVE TIME	OUTER ITER	INNER ITER	CONSTR. TIME	SOLVE TIME
h = 16	23	1	0.02	0.19	6	1	0.02	0.08
32	23	2	0.05	0.74	7	2	0.09	0.46
64	23	3	0.25	5.27	7	3	0.50	2.11
128	24	4	1.67	33.4	9	5	2.46	10.6
256	28	6	11.4	186	10	9	13.7	75.5

Table 6.7: Number of outer iterations in FGMRES (accuracy 10^{-6}), average number of inner iterations in GMRES (accuracy 10^{-3} and CPU-times required to solve the Oseen equations for $\nu = 1/80$ for different grid sizes. In all cases $\omega = 1$. Subsystems are solved using MATLAB's LUINC factorization with drop tolerance $1 \cdot 10^{-4}$ (KLW) AND $3 \cdot 10^{-4}$ (AC) AS PRECONDITIONER.

involve six variables, namely the flow velocity in longitudinal, latitudinal and vertical (depth) direction (u,v,w), pressure (p), temperature (T) and salinity (S). The adverb *thermohaline* means that the flow is driven by temperature and heat fluxes. We have six conservation laws: conservation of momentum in the three directions and conservation of mass, heat and salinity. For a detailed description of the equations and the precise parameter setting see [86].

The longitudinal and latitudinal momentum equation are similar, the same holds for the conservation of salinity and heat, therefore a natural clustering of variables is (u,v) and (S,T). Because of the Coriolis force that is involved we have to discretize on a spherical Lorenz grid, which means that we have a B-grid in the horizontal (longitudinal and latitudinal) direction and a C-grid in the vertical (depth) direction. A motivation for this special grid is given in [87]. The discretized system has the following structure

$$\begin{pmatrix} A_{uv} & 0 & G_{uv} & 0 \\ 0 & 0 & G_w & B_{ST} \\ D_{uv} & D_w & 0 & 0 \\ B_{uv} & B_w & 0 & A_{ST} \end{pmatrix} \begin{pmatrix} x_{uv} \\ x_w \\ x_p \\ x_{ST} \end{pmatrix} = \begin{pmatrix} b_{uv} \\ b_w \\ b_p \\ b_{ST} \end{pmatrix}. \tag{6.28}$$

The four rows represent respectively conservation of momentum in horizontal direction, conservation of momentum in vertical direction (which simplifies to the hydrostatic pressure equation), conservation of mass and conservation of heat and salinity.

The submatrix A_{uv} consists of two convection-diffusion equations that are coupled by the Coriolis force. The coupling is skew-symmetric, which makes the problem more difficult. Another cause of troubles is the zero diagonal block in the vertical momentum equation. This block is empty because the equation is dominated by the balance between gravity and pressure.

We designed a preconditioner for the equation. It is based on depth integration of the pressure, a transformation of the continuity equation and an asymmetric rearrangement of rows and columns. See Chapter 7 for the derivation of the preconditioner and for details on the implementation. In this chapter it is enough to know that at some point we have to solve a depth-averaged saddle point problem. Let M_{uv} denote the depth averaging operator for the velocities

and M_p the depth averaging operator for the pressure. We can then define the depth average matrices $A = M_{uv}A_{uv}M_{uv}^T$, $B_1 = M_{uv}G_{uv}M_p^T$ and $B_2 = M_pD_{uv}M_{uv}^T$. Equation (6.28) contains a saddle point problem that is revealed if we remove the second and fourth column and row. Depth averaging of that saddle point problem results in a new saddle point problem and that is the one we will consider.

$$\begin{pmatrix} A & B_1 \\ B_2 & 0 \end{pmatrix} \begin{pmatrix} \bar{x}_{uv} \\ \bar{x}_p \end{pmatrix} = \begin{pmatrix} \bar{b}_{uv} \\ \bar{b}_p \end{pmatrix}. \tag{6.29}$$

One of the properties of the resulting saddle point equation is that A is a convection-diffusion equation involving a Coriolis force. In the appendix we show via Fourier analysis that the Coriolis force does not necessarily destroy the property that the Schur complement $C = B_2A^{-1}B_1$ is independent of the mesh, though the condition number may be substantially larger.

As a test problem we isolate a saddle point problem from a developed flow in a rectangular box in the North Atlantic (286-350 degrees longitude, 10-74 degrees latitude). With a developed flow we mean that wind, temperature, salinity and Coriolis force play an important role in the solution. In other words we deal with a real ocean flow problem. We extract saddle point problems for varying mesh sizes. The problem has a horizontal Ekman number

$$E_H = \frac{A_H}{2\Omega r_0^2} = 4 \cdot 10^{-5}.$$

The nondimensional Ekman number is the ratio of the frictional force per unit mass to the Coriolis acceleration, and a Reynolds number $Re = 2.5$. The low Reynolds number is because of the relative low resolution, which forces us to use a large A_H, the horizontal viscosity parameter.

In Table 6.8 one finds the number of iterations for some of the preconditioners we treated in this chapter. First of all we have to remark that we do not show the results for the ES- and WS-preconditioner. Both are not able to produce an acceptable preconditioner. The maximum number of iterations (300) in BICGSTAB is exceeded without getting near the desired accuracy. Furthermore the KLW-preconditioner cannot be used. It is not defined for this type of problem. Because of the presence of a dominant skew-symmetric Coriolis force it is no longer clear what kind of A_p we should choose such that Equation (6.13) is a good approximation.

If we apply the SIMPLE and SIMPLER preconditioners to the saddle point problem from ocean circulation we get very bad results: the maximum number of iterations is exceeded before we reach the desired accuracy. However we can improve the performance of the preconditioners. As we pointed out before, the Coriolis force is very important in the equations. It is essentially a coupling between u and v and after discretization it will create an off-diagonal entry. The SIMPLE(R) preconditioner uses the diagonal of A and therefore it does not see the important Coriolis force. We can tackle this problem by a simple modification. Assume that we order u and v such that the horizontal velocities that belong to the same grid point are clustered. If we now take the 2×2-block-diagonal of A instead of the normal diagonal, we include the discretization of the Coriolis force. We will call the resulting preconditioners modified SIMPLE(R) or MSI and MSR for short.

METHOD	MSI	MSR	GD	GD	GD	AC	AC	AC
ω	-	-	16	64	256	16	64	256
n = 768	16	7	67	30	11	33	12	5
3072	33	23	96	26	13	34	13	5
6912	78	34	78	30	12	33	12	5
12288	116	47	80	24	12	31	13	5
19200	158	67	45	21	9	23	9	4

Table 6.8: Average number of iterations in BICGSTAB (accuracy 10^{-6}) for the modified SIMPLE(R), grad-div and artificial compressibility preconditioner for a depth-averaged saddle point problem from ocean circulation. Subsystems are solved exactly

The preconditioner proposed in [57] for the rotation form of the Navier-Stokes equations could be applied to the ocean problem as well. It is a variant of the KLW-preconditioner that we described in Section 6.2.4. It uses a different approximation to the Schur complement that involves a 2×2-block-diagonal approximation to A similar to the one we use in the MSI- and MSR-preconditioner. The numerical results we obtain for the preconditioner in [57] are similar to that of MSI, what is not a surprise if one takes in account the similarity between Theorems 6.2.1 and 6.2.5. Because these preconditioners are so closely related, we only show the numerical results for the MSI-preconditioner.

In Table 6.8 we compare the results for different preconditioners. The problem size in the table is the size of the depth-averaged saddle point problem of Equation (6.29). The reader should keep in mind that the underlying ocean problem as described in (6.28) is at least 32 times bigger. The solution of the full ocean equations requires a number of solves of the depth-averaged saddle point problem.

We have to choose a relatively large value for ω to get acceptable results for the grad-div and artificial compressibility preconditioner. This is due to the Coriolis force and bad scaling of the original matrix. For the values shown in the table, the number of iterations seems to depend only slightly on the problem size. For the modified SIMPLE and SIMPLER preconditioners the dependence is a lot stronger.

Finally we show in Table 6.9 the CPU-times for the preconditioners on the largest saddle point problem if we solve the subsystems with only one application of MATLAB's LUINC factorization. It seems that the grad-div and artificial compressibility preconditioners are more sensitive to the replacement of exact solves by approximations than the modified SIMPLE and SIMPLER method. The number of iterations using the grad-div preconditioner is even five times bigger than in the case of exact solution of subsystems. This is possibly due to the asymmetry of the saddle point problem. The artificial compressibility preconditioner is the fastest in this case, although the modified SIMPLE and SIMPLER preconditioner perform quite well too.

METHOD	MSI	MSR	GD	AC
VALUE OF ω	-	-	16	16
ITERATIONS	155	83	254	40
CONSTRUCTION TIME	1.0	1.0	1.9	1.9
SOLVE TIME	41	36	152	24

Table 6.9: Number of iterations in FGMRES (accuracy 10^{-6}) and CPU-times required for the solution of the largest depth-averaged saddle point problem from ocean circulation ($n = 19200$). Subsystems are solved with GMRES (accuracy 10^{-3}) using MATLABS's LUINC with drop tolerance $3e-4$ as preconditioner.

6.5 Summary and discussion

In this chapter we treated two preconditioners for the saddle point problem: one based on the augmented Lagrangian and one on artificial compressibility. Eigenvalue analysis shows that small condition numbers are achievable. This analysis is supported by the numerical experiments for the Stokes equation, the Oseen equation and a saddle point problem from an ocean model. In all cases the artificial compressibility preconditioner performs better than the preconditioners SIMPLE(R), ES, WS and KLW.

One might consider the presence of the parameter ω in the preconditioners as a disadvantage. However the advantage of this parameter is that we can shift work from inner to outer iterations and back. If we increase the value of ω the condition number of the preconditioned matrix will be closer to 1 and the number of outer iterations will decrease. Meanwhile the grad-div added matrix will be more difficult to solve so the number of inner iterations will increase. The parameter should be chosen such that there is a balance in work in the inner and outer iteration. In general if the entries in A and B are scaled such that they are of the same magnitude, $\omega = \mathcal{O}(1)$ works fine. Our experience is that near this value the increase or decrease of ω with a factor of 2 will shift the amount of work, but the overall CPU-time will be almost the same. Hence, fortunately the choice is not very critical.

As we mentioned in Section 6.3.4 the question of a robust preconditioner for grad-div added matrices is not fully solved. We sketched several possibilities. However the numerical experiments show that in practice we can apply the methods for A to $A + \omega BB^T$ as well, at least for modest values of ω. Nevertheless the solution of these systems needs more study especially for larger values of ω.

One of the advantages of the described preconditioners is that they perform well on the three different test problems without concern. The eigenvalue analysis shows that the preconditioners will always perform well on a saddle point problem that has the structure of (6.1) as long as we choose ω large enough. This does certainly not hold for the preconditioners from literature that we described in this chapter. The KLW-preconditioner is only defined for the Oseen equations; the WS- and ES-preconditioners perform bad on Oseen and ocean equations; the SIMPLE(R) preconditioners do not perform well on the Oseen equations and we have to modify the methods before we can apply them to the ocean equations.

Both eigenvalue analysis and numerical experiments show that the grad-div and artificial compressibility preconditioners might be good alternatives for existing preconditioners in the solution of saddle point problems in fluid flows.

6.6 Appendix: Fourier analysis for Stokes with Coriolis

In Section 6.4.3 we stated that the eigenvalues of the Schur complement $C = B^T A^{-1} B$ in case of the saddle point problem from the ocean equations are independent of the mesh size. Here we will motivate that via Fourier analysis on the Stokes equation with a Coriolis force, because in the ocean flow equations the diffusion terms dominate the convection terms.

We start with the continuous case. The Stokes equation with a Coriolis force yields a saddle point problem of the form (6.1) where A is a discrete version of

$$\begin{pmatrix} \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} & \Omega \\ -\Omega & \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \end{pmatrix}.$$

Here Ω represents the ratio of the Coriolis force and the viscosity; we assume Ω to be constant. The matrix B is the discretization of

$$\begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix}.$$

If we define

$$\begin{pmatrix} u \\ v \\ p \end{pmatrix} = \begin{pmatrix} u_0 \\ v_0 \\ p_0 \end{pmatrix} e^{if_1 x + if_2 y},$$

the symbols of the continuous form of A and B become

$$\hat{A} = \begin{pmatrix} f_1^2 + f_2^2 & \Omega \\ -\Omega & f_1^2 + f_2^2 \end{pmatrix} \text{ and } \hat{B} = \begin{pmatrix} if_1 \\ if_2 \end{pmatrix}.$$

So the inverse of $\hat{A}$ is

$$\hat{A}^{-1} = \frac{1}{(f_1^2 + f_2^2)^2 + \Omega^2} \begin{pmatrix} f_1^2 + f_2^2 & -\Omega \\ \Omega & f_1^2 + f_2^2 \end{pmatrix}$$

Hence $\hat{C} = \hat{B}^* \hat{A}^{-1} \hat{B}$ becomes (split $\hat{A}^{-1}$ in symmetric and skew symmetric part):

$$\hat{C} = \frac{(f_1^2 + f_2^2)^2}{(f_1^2 + f_2^2)^2 + \Omega^2}$$

The largest eigenvalue is bounded by 1, and the smallest by $f_{\min}^4/\Omega^2$. The minimum frequency $f_{\min}$ is determined by the domain and the boundary conditions where we exclude the frequency zero, which yields a smallest eigenvalue zero of $\hat{C}$. That is not relevant if the right-hand side is in the image of the operator. So the condition number of the continuous variant of C based on Fourier analysis is bounded by $\Omega^2 / f_{\min}^4$. Hence, it is finite for all $f_1, f_2 > f_{\min}$.

For the discrete case on a B-grid (for a motivation see [87]) the symbols of A and B are

$$\tilde{A} = \begin{pmatrix} \frac{4}{h^2}\left(\sin^2(\frac{f_1 h}{2}) + \sin^2(\frac{f_2 h}{2})\right) & -\Omega \\ \Omega & \frac{4}{h^2}\left(\sin^2(\frac{f_1 h}{2}) + \sin^2(\frac{f_2 h}{2})\right) \end{pmatrix}, \tilde{B} = \begin{pmatrix} \frac{2i}{h}\cos(\frac{f_2 h}{2})\sin(\frac{f_1 h}{2}) \\ \frac{2i}{h}\cos(\frac{f_1 h}{2})\sin(\frac{f_2 h}{2}) \end{pmatrix},$$

which leads in an analogous way to the symbol

$$\tilde{C} = \frac{\left(\sin^2(\frac{f_1 h}{2}) + \sin^2(\frac{f_2 h}{2})\right)\left(\cos^2(\frac{f_2 h}{2})\sin^2(\frac{f_1 h}{2}) + \cos^2(\frac{f_1 h}{2})\sin^2(\frac{f_2 h}{2})\right)}{\left(\sin^2(\frac{f_1 h}{2}) + \sin^2(\frac{f_2 h}{2})\right)^2 + \Omega^2}.$$

In this case the largest eigenvalue is bounded by $\frac{4}{4+\Omega^2}$ which is by itself less than one. For the smallest eigenvalue we have two cases from which we have to take the minimum. The first one is the same as that of the continuous case and the second occurs if the cosines in the numerator are close to zero. In any case the minimum eigenvalue is at least the ratio of the minimum of the numerator divided by the maximum of the denominator. The latter is simply $4+\Omega^2$ and an elementary analysis shows that the sought minimum of the numerator occurs if $f_1 = f_2$ and f_1 is closest to π/h (i.e. the highest frequency that can be represented on the grid). The numerator will then have a value approximately $(\pi - f_{max}h)^2$, hence the condition number in the discrete case is

$$\max(\Omega^2/f_{\min}^4, 4/(\pi - f_{max}h)^2).$$

Now, if the mesh size decreases f_{max} will tend to π/h, we exclude here the sawtooth Fourier component which gives a smallest eigenvalue zero. Hence, the second part in the max expression will dominate the first if the mesh size becomes small. However, in ocean models we have quite large mesh sizes and currently we do not encounter problems due to a dominating second part.

Chapter 7

A tailored solver for bifurcation analysis of ocean-climate models

7.1 Introduction

In the previous chapters we payed much attention to the saddle point problem. We discussed a direct solver (Chapter 4), preconditioning via incomplete LU factorization (Chapter 5) and preconditioning with block-preconditioners (Chapter 6). In this chapter we will return to the main issue, namely the solution of large linear systems from the model as presented in Chapter 2.

In Section 2.4.2 we showed that the Jacobian matrix of the discrete ocean equations (Φ) has the following structure

$$\begin{bmatrix} \mathbf{A}_{uv} & \mathbf{E}_{uv} & \mathbf{G}_{uv} & 0 \\ 0 & 0 & \mathbf{G}_{w} & \mathbf{B}_{TS} \\ \mathbf{D}_{uv} & \mathbf{D}_{w} & 0 & 0 \\ \mathbf{B}_{uv} & \mathbf{B}_{w} & 0 & \mathbf{A}_{TS} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{uv} \\ \mathbf{u}_{w} \\ \mathbf{u}_{p} \\ \mathbf{u}_{TS} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{uv} \\ \mathbf{b}_{w} \\ \mathbf{b}_{p} \\ \mathbf{b}_{TS} \end{bmatrix}. \tag{7.1}$$

In previous versions of THCM (the thermohaline continuation model) this system was solved with MRILU, a multilevel renumbering incomplete LU factorization method described in [10]. MRILU was not applied to the matrix with the block ordering as in (7.1) but to a matrix where the variables were interleaved, that is, the variables (u,v,w,p,T,S) that belong to the same cell were put after each other in the ordering. The resulting system was manipulated by MRILU with blocksize 6, i.e. the 6 variables were treated as one single variable. The major disadvantages of MRILU are the memory requirements and the construction time. Both are related to the large amount of fill (the nonzeros) in the final factorization.

We cannot apply the methods of the previous chapters blindly to (7.1), simply because the matrix is not of saddle point type. However based on the conclusions of Chapter 5 we can say that it is unlikely that the incorporation of fill-reducing orderings in MRILU or another incomplete LU factorization method will result in a good preconditioner with low fill. It was not able to do so for the Stokes problem, let alone for the ocean system. Chapter 6 showed us that approaches that exploit the structure of the saddle point problem, can result in very efficient

preconditioners. This motivated us to design a preconditioner for the matrix in (7.1) based on its structure.

We carefully studied the submatrices in (7.1) and, based on their properties, we constructed a preconditioner that subsequently solves different variables. After the development of the preconditioner we noted that there was a striking similarity with a technique well known and widely applied in explicit time integration methods for ocean flows: *barotropic-baroclinic time splitting*, see e.g. [42]. This technique exploit the fact that fast external gravity waves are approximately independent of depth. Hence the time-dependent equations are solved in the following way:

(1) apply depth-averaging to momentum and continuity equations (2.4a-b,d);
(2) solve the depth averaged horizontal velocity field using the barotropic equations (two-dimensional);
(3) construct full pressure field (p) and horizontal velocity field (u,v) using the baroclinic equations;
(4) solve vertical velocity field (w) via the continuity equation;
(5) solve tracer equations (T,S).

Both depth-averaging and the typical ordering of the solution of variables plays an important role in our preconditioner. The techniques used in explicit time stepping appear to be helpful in the design of a preconditioner for the Jacobian matrix that occurs in the implicit time stepping methods. Note that the Jacobian that occurs in continuation of steady states is nothing but an extreme case ($\Delta t = \infty$) of the Jacobian in implicit time stepping (see Section 2.6).

In this chapter we describe the tailored solver and motivate the choices we made in the design. To be able to exploit the structure of the equations we will rewrite the system in Section 7.2. The rewritten system will serve as a starting point for the construction of an (incomplete) block LU factorization. In Section 7.4 we will discuss the practical implementation of the preconditioner and the solution of the subsystems.

The performance of the preconditioner is not discussed in this chapter. We leave that for Chapter 8, where we solve several large scale ocean flows.

7.2 Rewriting the equations to ease the solution process

Before we describe the preconditioner for (7.1) we first rewrite the system in four subsequent steps, we (i) transform the pressure, (ii) transform the continuity equation, (iii) rename some submatrices and finally (iv) we rearrange columns and rows. The aim of all these operations is to get a system that is a better starting point for the development of a preconditioner. We remark that for the construction of the preconditioner we ignore the block $\mathbf{E}_{uv}$ in (7.1), because the entries in this block are very small compared to entries in the rest of the matrix. There is a physical motivation for this as well. The terms $w\,u_z$ and $w\,v_z$ are small relative to the

Coriolis force and the pressure gradient in the momentum equations for u (Equation (2.4a)) and v (Equation (2.4b)) respectively.

The difficulties in solving Equation (7.1) are partially caused by the zero blocks on the diagonal. We could get rid of these block, if it was allowed to interchange the (block) rows 2 and 3, which is not the case, because the matrices $\mathbf{G}_w$ and $\mathbf{D}_\mathbf{w}$ are not square ($d_w < d_p$). However, after a smart transformation of the pressure and the continuity equation, we can split the matrices in two subblocks: a square invertible matrix and a zero matrix. The square and invertible submatrices then allow to interchange columns and rows. In the next paragraph we will pay attention to the transformation of the pressure.

Step 1: transform the pressure

In the continuous formulation the hydrostatic pressure equation, see Equation (2.4c), contains the term p_z. Note that we can decompose the pressure in $p = \overline{p} + \tilde{p}$, where $\overline{p}(\phi,\theta) = \int_{-h}^{0} p(\phi,\theta,z)dz$ is a depth-averaged pressure, that vanishes in the hydrostatic pressure equation, because it is in the kernel of the operator $\frac{\partial}{\partial z}$. From this it follows that $\int_{-h}^{0} \tilde{p}(\phi,\theta,z)dz = 0$. The consequence of this decomposition is that the problem $f_z = g$ in the space of all functions with $\int_{-h}^{0} f\, dz = 0$, is now uniquely solvable.

We can do something similar on the discrete equations. As we pointed out in Section 2.4 the matrix $\mathbf{G}_w$ has dimensions $d_w \times d_p$ and because $d_w < d_p$ it is not square. The matrix has full rank, so the kernel has dimension $d_p - d_w$. Because of the structured grid and the fact that $\mathbf{G}_w$ is a discrete gradient operator, it is quite easy to construct an orthonormal basis for this kernel. Given such a basis we can build the operator $\mathbf{M}_1$, who's rows are precisely these orthonormal basis vectors. Consequently $\mathbf{G}_w\mathbf{M}_1^T = 0$ and $\mathbf{M}_1\mathbf{M}_1^T = \mathbf{I}_{\overline{p}}$. The operator $\mathbf{M}_1$ acts on the pressure space and is up to a factor a depth-averaging of the pressure field.

We now define the transformation matrix

$$\mathbf{T}_1 = \begin{bmatrix} \mathbf{I}_w & \mathbf{M}_1^T \\ 0 & \end{bmatrix},$$

which is a square matrix, because it is the matrix $\mathbf{M}_1^T$ with dimension $d_p \times (d_p - d_w)$ put together with a $d_w \times d_w$ identity matrix and a $(d_p - d_w) \times d_w$ zero block. If we choose an appropriate ordering for the variables $\mathbf{u}_p$ it is guaranteed that the matrix is invertible. In practice it suffices to order the pressure nodes such that all nodes at the surface are numbered last. Each surface node corresponds to one singular vector of $\mathbf{G}_w$.

With this $\mathbf{T}_1$ we can rewrite the pressure field:

$$\mathbf{u}_p = \mathbf{T}_1\mathbf{u}_{\hat{p}} = \begin{bmatrix} \mathbf{I}_w & \mathbf{M}_1^T \\ 0 & \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\tilde{p}} \\ \mathbf{u}_{\overline{p}} \end{bmatrix}. \tag{7.2}$$

If we use this equation in system (7.1) we get two transformed gradient matrices, namely

$$\mathbf{G}_{uv}\mathbf{T}_1 = \begin{bmatrix} \tilde{\mathbf{G}}_{uv} & \mathbf{G}_{uv}\mathbf{M}_1^T \end{bmatrix} \text{ and } \mathbf{G}_w\mathbf{T}_1 = \begin{bmatrix} \tilde{\mathbf{G}}_w & \mathbf{G}_w\mathbf{M}_1^T \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{G}}_w & 0 \end{bmatrix}, \tag{7.3}$$

where the last equality holds because of the definition of $\mathbf{M}_1$. The matrices $\tilde{\mathbf{G}}_{uv}$ and $\tilde{\mathbf{G}}_w$ contain the first d_w columns of $\mathbf{G}_{uv}$ and $\mathbf{G}_w$ respectively. The matrix $\tilde{\mathbf{G}}_w$ has d_w rows as well, so it is square. Because $\mathbf{G}_w$ and $\tilde{\mathbf{G}}_w$ have the same rank the latter matrix necessarily is invertible.

Step 2: transform the continuity equation

Since $w_z = f$ with $w(0) = w(-h) = 0$ is overdetermined, there is no solution unless $\int_{-h}^{0} f\, dz = 0$. The latter is obtained from the depth integration of the continuity equation, see Equation (2.4d), where w vanishes because of the boundary conditions. This gives an equation for the horizontal velocities. With the horizontal velocities satisfying this equation, one can solve w uniquely and leave out even one of the boundary conditions.

We can perform this depth integration as well on the discrete equations. In analogy to step 1 we use the operator $\mathbf{M}_2$, who's rows form an orthonormal basis for the kernel of $\mathbf{D}_w^T$. So each row of $\mathbf{M}_2$ is a left-singular vector of $\mathbf{D}_w$. Consequently $\mathbf{M}_2\mathbf{D}_w = 0$ and $\mathbf{M}_2\mathbf{M}_2^T = \mathbf{I}_{\overline{p}}$. With $\mathbf{M}_2$ we define the following transformation matrix

$$\mathbf{T}_2 = \begin{bmatrix} \mathbf{I}_w & 0 \\ \mathbf{M}_2 & \end{bmatrix}.$$

The matrix is square and invertible.
Note that if $\mathbf{D}_w^T = \mathbf{G}_w$, we would have $\mathbf{M}_1 = \mathbf{M}_2$ and $\mathbf{T}_1 = \mathbf{T}_2^T$. However equality of the discrete divergence and gradient matrices only holds for equidistant grids. In case of a stretched grid the matrices $\mathbf{D}_w^T$ and $\mathbf{G}_w$ need some diagonal scaling to make them equal. The last property is important for stability.

We apply $\mathbf{T}_2$ to the discrete equation of conservation mass, that is the third block-row in (7.1). We get two transformed divergence matrices

$$\mathbf{T}_2\mathbf{D}_{uv} = \begin{bmatrix} \tilde{\mathbf{D}}_{uv} \\ \mathbf{M}_2\mathbf{D}_{uv} \end{bmatrix} \text{ and } \mathbf{T}_2\mathbf{D}_w = \begin{bmatrix} \tilde{\mathbf{D}}_w \\ \mathbf{M}_2\mathbf{D}_w \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{D}}_w \\ 0 \end{bmatrix}. \tag{7.4}$$

Where the matrices $\tilde{\mathbf{D}}_{uv}$ and $\tilde{\mathbf{D}}_w$ contain the first d_w rows of $\mathbf{D}_{uv}$ and $\mathbf{D}_w$ respectively. The last matrix has d_w columns as well so $\tilde{\mathbf{D}}_w$ will be square and invertible. We get the zero block in the last equality because of the definition of $\mathbf{D}_w$. Furthermore in the right-hand-side we get

$$\mathbf{T}_2\mathbf{b}_p = \begin{bmatrix} \mathbf{b}_{\tilde{p}} \\ \mathbf{b}_{\overline{p}} \end{bmatrix},$$

where $\mathbf{b}_{\tilde{p}} = [\mathbf{I}_w \;\; 0]\mathbf{b}_p$ and $\mathbf{b}_{\overline{p}} = \mathbf{M}_2\mathbf{b}_p$.

Step 3: isolate saddle point problem and rename subblocks

If we put the transformations together we get the following system

$$\begin{bmatrix} \mathbf{A}_{uv} & 0 & \tilde{\mathbf{G}}_{uv} & \mathbf{G}_{uv}\mathbf{M}_1^T & 0 \\ 0 & 0 & \tilde{\mathbf{G}}_w & 0 & \mathbf{B}_{TS} \\ \tilde{\mathbf{D}}_{uv} & \tilde{\mathbf{D}}_w & 0 & 0 & 0 \\ \mathbf{M}_2\mathbf{D}_{uv} & 0 & 0 & 0 & 0 \\ \mathbf{B}_{uv} & \mathbf{B}_w & 0 & 0 & \mathbf{A}_{TS} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{uv} \\ \mathbf{u}_w \\ \mathbf{u}_{\tilde{p}} \\ \mathbf{u}_{\overline{p}} \\ \mathbf{u}_{TS} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{uv} \\ \mathbf{b}_w \\ \mathbf{b}_{\tilde{p}} \\ \mathbf{b}_{\overline{p}} \\ \mathbf{b}_{TS} \end{bmatrix}. \tag{7.5}$$

To simplify the matrix we define

$$\mathbf{K}_{uv\overline{p}} = \begin{bmatrix} \mathbf{A}_{uv} & \mathbf{G}_{uv}\mathbf{M}_1^T \\ \mathbf{M}_2\mathbf{D}_{uv} & 0 \end{bmatrix}, \tag{7.6}$$

which is a saddle point matrix involving the full velocity field and the depth-averaged pressure field. $\mathbf{K}_{uv\overline{p}}$ is a submatrix of (7.5) (involving block-row and -column 1 and 4) and we want to isolate this system. To do so we have to define

$$\hat{\mathbf{B}}_{uv} = \begin{bmatrix} \mathbf{B}_{uv} & 0 \end{bmatrix}, \hat{\mathbf{D}}_{uv} = \begin{bmatrix} \tilde{\mathbf{D}}_{uv} & 0 \end{bmatrix}, \tag{7.7}$$

$$\hat{\mathbf{G}}_{uv} = \begin{bmatrix} \tilde{\mathbf{G}}_{uv} \\ 0 \end{bmatrix}, \mathbf{u}_{uv\overline{p}} = \begin{bmatrix} \mathbf{u}_{uv} \\ \mathbf{u}_{\overline{p}} \end{bmatrix} \text{ and } \mathbf{b}_{uv\overline{p}} = \begin{bmatrix} \mathbf{b}_{uv} \\ \mathbf{b}_{\overline{p}} \end{bmatrix}. \tag{7.8}$$

Furthermore we introduce $\mathbf{A}_p = \tilde{\mathbf{G}}_w$ and $\mathbf{A}_w = \tilde{\mathbf{D}}_w$, to reflect that the matrices are square and invertible. After some rearrangement we get the system

$$\begin{bmatrix} \mathbf{K}_{uv\overline{p}} & 0 & \hat{\mathbf{G}}_{uv} & 0 \\ 0 & 0 & \mathbf{A}_p & \mathbf{B}_{TS} \\ \hat{\mathbf{D}}_{uv} & \mathbf{A}_w & 0 & 0 \\ \hat{\mathbf{B}}_{uv} & \mathbf{B}_w & 0 & \mathbf{A}_{TS} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{uv\overline{p}} \\ \mathbf{u}_w \\ \mathbf{u}_{\tilde{p}} \\ \mathbf{u}_{TS} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{uv\overline{p}} \\ \mathbf{b}_w \\ \mathbf{b}_{\tilde{p}} \\ \mathbf{b}_{TS} \end{bmatrix}. \tag{7.9}$$

This system looks the same as system (7.1). The only difference seems the names of the blocks and some hats and tildes. So what is the gain of the transformations and redefinition of all the subblocks? Well first of all the systems (7.1) and (7.9) may look similar, but they are different. The most important difference is that the blocks $\mathbf{A}_w$ and $\mathbf{A}_p$ in (7.9) are square and invertible, whereas $\mathbf{G}_w$ and $\mathbf{D}_w$ in (7.1) have a different number of columns and rows. Because these two matrices now are of the same size and invertible we can swap rows or columns 2 and 3, such that the blocks move to the diagonal. Doing so we reduce the number of zeros on the diagonal dramatically. The second important difference is that by the transformation, rearrangement and redefinition of subblocks, we moved part of the discrete gradient and the divergence matrices to the heading block $\mathbf{K}_{uv\overline{p}}$ that became a saddle point matrix instead of the simple discrete convection-diffusion equation with Coriolis $\mathbf{A}_{uv}$ in (7.1). As we saw in Chapter 6 in general saddle point problems are more difficult to solve, but fortunately in this case we have good preconditioners available.

Step 4: rearrange columns and rows

Note that in system (7.9) there is a cyclic dependency of the different types of unknowns. If the pressure field $\mathbf{u}_{\tilde{p}}$ is given, we can use the first (block) row to solve the horizontal velocity and the depth-averaged pressure $\mathbf{u}_{uv\overline{p}}$. Given these two we can compute the vertical velocity $\mathbf{u}_w$ with the third row. Then we can use the last row to solve $\mathbf{u}_{TS}$, which on its turn can be used to compute a new $\mathbf{u}_{\tilde{p}}$ with the second row. In a scheme this dependency is

$$\mathbf{u}_{\tilde{p}} \Rightarrow \mathbf{u}_{uv\overline{p}} \Rightarrow \mathbf{u}_w \Rightarrow \mathbf{u}_{TS} \Rightarrow \mathbf{u}_{\tilde{p}}.$$

This dependency can be made more explicit if we asymmetrically rearrange the columns (variables) and rows (equations) in the matrix. Numbering the rows and columns from 1 to 4, we use for the rows the permutation $q_r = (2,1,3,4)$ and for the columns $q_c = (3,1,2,4)$. This rearrangement gives a system with an almost lower block triangular matrix

$$\begin{bmatrix} \mathbf{A}_p & 0 & 0 & \mathbf{B}_{TS} \\ \hat{\mathbf{G}}_{uv} & \mathbf{K}_{uv\overline{p}} & 0 & 0 \\ 0 & \hat{\mathbf{D}}_{uv} & \mathbf{A}_w & 0 \\ 0 & \hat{\mathbf{B}}_{uv} & \mathbf{B}_w & \mathbf{A}_{TS} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\tilde{p}} \\ \mathbf{u}_{uv\overline{p}} \\ \mathbf{u}_w \\ \mathbf{u}_{TS} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_w \\ \mathbf{b}_{uv\overline{p}} \\ \mathbf{b}_{\tilde{p}} \\ \mathbf{b}_{TS} \end{bmatrix}. \tag{7.10}$$

We will call this transformed and permuted Jacobian matrix $\hat{\Phi}$.

If we use wind forcing only, the blocks $\hat{\mathbf{B}}_{uv}$ and $\mathbf{B}_w$ are empty. Then we can solve $\mathbf{u}_{TS}$ first, followed by $\mathbf{u}_{\tilde{p}}$, $\mathbf{u}_{uv\overline{p}}$ and $\mathbf{u}_w$.

Remark 7.2.1. There are two other permutations of (7.9) that give an almost lower block triangular matrix. If we use $q_r = (1,3,4,2)$ and $q_c = (1,2,4,3)$, the matrix $\mathbf{A}_p$ becomes the last diagonal block. Then $\hat{\mathbf{G}}_{uv}$ is the only upper diagonal block. The last possibility is $q_r = (3,4,2,1)$ and $q_c = (2,4,3,1)$, which gives $\mathbf{K}_{uv\overline{p}}$ as last diagonal block and two nonzero upper diagonal blocks, namely $\hat{\mathbf{D}}_{uv}$ and $\hat{\mathbf{B}}_{uv}$. The permutation that we chose corresponds best to the ordering of variables in barotropic-baroclinic splitting, that we discussed in the introduction. We will further motivate this choice in Remark 7.3.1.

Note that so far we did not essentially change the system; equations (7.1) and (7.10) have the same solution, although they look quite different. The most important advantage of (7.10) is that the amount of zeros on the diagonal has reduced dramatically. (7.10) has only one zero block (hidden in the heading block) where (7.1) has two. Furthermore, the zero block in the rewritten system is rather small because it is part of the depth-averaged continuity equation. The rewritten system appears to be more convenient as a starting point for the development of a preconditioner than (7.1).

7.3 Block (incomplete) LU factorization

From $\hat{\Phi}$ in Equation (7.10) we can construct the following block LU factorization

$$\mathbf{L}_{\hat{\Phi}}\mathbf{U}_{\hat{\Phi}} = \begin{bmatrix} \mathbf{A}_p & 0 & 0 & 0 \\ \hat{\mathbf{G}}_{uv} & \mathbf{K}_{uv\overline{p}} & 0 & 0 \\ 0 & \hat{\mathbf{D}}_{uv} & \mathbf{A}_w & 0 \\ 0 & \hat{\mathbf{B}}_{uv} & \hat{\mathbf{B}}_w & \mathbf{S}_{TS} \end{bmatrix} \begin{bmatrix} \mathbf{I}_p & 0 & 0 & \mathbf{A}_p^{-1}\mathbf{B}_{TS} \\ 0 & \mathbf{I}_{uv+\overline{p}} & 0 & -\mathbf{K}_{uv\overline{p}}^{-1}\hat{\mathbf{G}}_{uv}\mathbf{A}_p^{-1}\mathbf{B}_{TS} \\ 0 & 0 & \mathbf{I}_w & \mathbf{A}_w^{-1}\hat{\mathbf{D}}_{uv}\mathbf{K}_{uv\overline{p}}^{-1}\mathbf{G}_{uv}\mathbf{A}_p^{-1}\mathbf{B}_{TS} \\ 0 & 0 & 0 & \mathbf{I}_{TS} \end{bmatrix}, \quad (7.11)$$

where the Schur complement is given by

$$\mathbf{S}_{TS} = \mathbf{A}_{TS} + (\hat{\mathbf{B}}_{uv} - \hat{\mathbf{B}}_w\mathbf{A}_w^{-1}\hat{\mathbf{D}}_{uv})\mathbf{K}_{uv\overline{p}}^{-1}\hat{\mathbf{G}}_{uv}\mathbf{A}_p^{-1}\mathbf{B}_{TS}. \quad (7.12)$$

This factorization is exact. However in practice it is impossible to construct the Schur complement explicitly, because the matrix will be dense. Now there are two possibilities:

(i) Totally forget the Schur complement and simply use $\mathbf{S}_{TS} = \mathbf{A}_{TS}$. In that case it is of no use to apply the block $\mathbf{U}_{\hat{\Phi}}$. This is the same as ignoring the block $\mathbf{B}_{TS}$ in (7.10), which leads to a block Gauss-Seidel preconditioner, which we will refer to as BLOCK-GS.

(ii) We solve Equation (7.12) via an iterative procedure. The application of the Schur complement to a vector can be decomposed in a number of matrix-vector multiplications and system solves, which is relative cheap. The problem that remains is the search of a good preconditioner for $\mathbf{S}_{TS}$. This results in a kind of block incomplete LU factorization, which we will call the BLOCK-ILU preconditioner.

Remark 7.3.1. If we choose a different permutation of (7.9), as we mentioned in Remark 7.2.1, we get a different block LU factorization with a different Schur complement. Numerical experiments showed that in approach (i) the timing results for all permutation are close to each other, although the permutation that we chose seems slightly better. Within approach (ii) the differences in timing results are much bigger. In that case iterative solution of the Schur complement $\mathbf{S}_{TS}$ gives the best results.

7.4 The implementation of the preconditioner

The application of the factorization (7.11) as preconditioner requires the solution of two equations, one with $\mathbf{L}_{\hat{\Phi}}$ and one with $\mathbf{U}_{\hat{\Phi}}$. The solution of both equations can be decomposed in matrix-vector multiplications and the solution of systems with the diagonal blocks $\mathbf{A}_p, \mathbf{A}_w, \mathbf{K}_{uv\overline{p}}$ and $\mathbf{A}_{TS}$ or the Schur complement $\mathbf{S}_{TS}$. We will discuss the solution of these systems in this section.

7.4.1 Solving the system for the pressure

The matrix $\mathbf{A}_p = \tilde{\mathbf{G}}_w$ is a square and invertible submatrix of $\mathbf{G}_w$ as implicitly defined by Equation (7.3). $\mathbf{G}_w$ is the discretization of the term p_z in Equation (2.4c). Because of the B-grid (see Figure 2.3) and the fact that the hydrostatic pressure equation is discretized at the *w*-points, the matrix $\mathbf{G}_w$ is a bi-diagonal matrix. In fact it is even a gradient-type matrix as in Definition 4.1.2 in Chapter 4. With row scaling we can get the entries of $\mathbf{A}_p$, an invertible gradient-type matrix, in $\{-1,0,1\}$, which allows us to apply Lemma 4.4.3. We can conclude there exist row and column permutations such that $\mathbf{A}_p$ is upper diagonal. Consequently an equation with $\mathbf{A}_p$ can be solved exactly at low cost. Because we use structured grids it is easy to find the ordering such that $\mathbf{A}_p$ is upper diagonal.

7.4.2 Solving the system for the vertical velocity

For the matrix $\mathbf{A}_w = \hat{\mathbf{D}}_w$ we have a similar reasoning. It is a square and invertible submatrix of $\mathbf{D}_w$ as implicitly defined by Equation (7.4). The continuity equation (2.4d) is discretized at the position of *p* on a B-grid (see Figure 2.3), which causes $\hat{\mathbf{D}}_w^T = \mathbf{A}_w^T$ to be a gradient-type matrix. Via row scaling and Lemma 4.4.3 we conclude that $\mathbf{A}_p$ is a permutation of a lower diagonal matrix. So equations with $\mathbf{A}_p$ are easy to solve exactly.

7.4.3 Solving the saddle point problem

Unfortunately not all subsystems are as easy to solve as the previous two. The saddle point problem $\mathbf{K}_{uv\overline{p}}$ as defined in Equation (7.6) is much more difficult. To the reader this will not come as a surprise because all three previous chapters deal with the solution of saddle point problems. We will use the knowledge developed in those chapters here.

First of all let us recall the system that we want to solve:

$$\mathbf{K}_{uv\overline{p}}\mathbf{u}_{uv\overline{p}} = \begin{bmatrix} \mathbf{A}_{uv} & \mathbf{G}_{uv}\mathbf{M}_1^T \\ \mathbf{M}_2\mathbf{D}_{uv} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}_{uv} \\ \mathbf{u}_{\overline{p}} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{uv} \\ \mathbf{b}_{\overline{p}} \end{bmatrix}. \tag{7.13}$$

This saddle point system is a little weird, because it involves the full three dimensional field of horizontal velocities and the depth averaged pressure field, which is essentially two dimensional.

We will describe two ways to solve the equation

(a) Reduce the number of unknowns by depth-averaging the velocity field as well. Solve that system and use the equation $\mathbf{A}_{uv}\mathbf{u}_{uv} = \mathbf{b}_{uv} - \mathbf{G}_{uv}\mathbf{M}_1^T\mathbf{u}_{\overline{p}}$ to compute the full velocity field from the depth-averaged pressure.

(b) Solve the system at once.

Option (a) requires an operator for the depth averaging of the velocity field. Suppose the matrix $\mathbf{M}_3$ is that operator. It can be constructed such that the rows of the matrix are orthonormal: $\mathbf{M}_3\mathbf{M}_3^T = \mathbf{I}_{\overline{uv}}$. If we apply this operator to the velocity field and to the momentum

equations we get the following saddle point problem

$$\begin{bmatrix} \mathbf{A}_{\overline{uv}} & \mathbf{G}_{\overline{uv}} \\ \mathbf{D}_{\overline{uv}} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\overline{uv}} \\ \mathbf{u}_{\overline{p}} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{\overline{uv}} \\ \mathbf{b}_{\overline{p}} \end{bmatrix}, \tag{7.14}$$

where

$$\begin{aligned} \mathbf{A}_{\overline{uv}} &= \mathbf{M}_3 \mathbf{A}_{uv} \mathbf{M}_3^T, \\ \mathbf{G}_{\overline{uv}} &= \mathbf{M}_3 \mathbf{G}_{uv} \mathbf{M}_1^T, \\ \mathbf{D}_{\overline{uv}} &= \mathbf{M}_2 \mathbf{D}_{uv} \mathbf{M}_3^T. \end{aligned}$$

It is important to note that Equations (7.14) and (7.13) are essentially different: the solution for $\mathbf{u}_{\overline{p}}$ is not the same, but hopefully the approximation is good enough. By depth integration, we at least use information that is spread all over the velocity field. One could choose a different operator than $\mathbf{M}_3$, for example one that picks a single horizontal velocity field instead of averaging. Then the approximation becomes worse and the overall performance of the preconditioner deteriorates.

Note that the system (7.14) corresponds to the two-dimensional barotropic equations, that we mentioned in the introduction of this chapter.

The depth-averaged saddle point problem can be solved with a Krylov subspace method (see Section 3.2) using one of the preconditioners as described in Chapter 6. In Section 6.4.3, especially Table 6.8, we showed that for the depth-averaged equations the artificial compressibility preconditioner is a good candidate. The application of the artificial compressibility preconditioner requires the solution of the grad-div added matrix

$$\mathbf{A}_{\overline{uv}}^{GD} = \mathbf{A}_{\overline{uv}} + \omega \mathbf{G}_{\overline{uv}} \mathbf{D}_{\overline{uv}}, \tag{7.15}$$

where the parameter ω needs to be chosen quite large, due to bad scaling of the starting equations, which influences the size of the entries in the subblocks $\mathbf{A}_{uv}$, $\mathbf{G}_{uv}$ and $\mathbf{D}_{uv}$, and their depth-integrated counterparts. The modified simpler approach, that we described in Section 6.2.1 and 6.4.3, is a good alternative. As the method applies as well to the larger saddle point problem (7.13), we will discuss it in more detail at the end of this paragraph.

Note that the size of the depth averaged saddle point problem (7.14) is L (the number of cells in the depth) times smaller than the original saddle point problem. It is much cheaper to solve than the big saddle point system (7.13). The time to solve it is only a fraction of the time needed to solve the much bigger systems with $\mathbf{A}_{uv}$ and $\mathbf{S}_{TS}$. Therefore, the choice of the preconditioner is not a crucial factor.

Given $\mathbf{u}_{\overline{p}}$ we can compute $\mathbf{u}_{uv}$ from the equation $\mathbf{A}_{uv}\mathbf{u}_{uv} = \mathbf{b}_{uv} - \mathbf{G}_{uv}\mathbf{M}_1^T\mathbf{u}_{\overline{p}}$. This is again done with a Krylov subspace method involving a preconditioner for the matrix $\mathbf{A}_{uv}$.

The choice of $\mathbf{M}_3$ is an important factor if we want to apply this method successfully. If we have no bottom topography and an equidistant grid the choice for the depth-averaging operator is quite clear. The method we described here gives a good preconditioner for the saddle point

problem (7.13). However as soon as we introduce bottom topography or stretched grids, it is more difficult to construct the right $\mathbf{M}_3$. If we choose the wrong one, the results deteriorate quickly. Apparently the solutions $\mathbf{u}_{\overline{p}}$ of Equations (7.14) and (7.13) differ too much in that case. It might be possible to find a matrix $\mathbf{M}_3$ such that the saddle point problem (7.14) is a better approximation to the barotropic equations and the solutions $\mathbf{u}_{\overline{p}}$ of both saddle point problems are close to each other. A further study of the barotropic-baroclinic splitting, like in [42], could help to find the appropriate depth averaging operator for the horizontal momentum equations. So far we did not succeed to construct this matrix.

For the global ocean flows the bottom topography and a stretched grid are important, so we have to use option (b): we solve the saddle point system at once. Of the preconditioners that we described in the previous section, there is only one that is useful as preconditioner for the saddle point system (7.13): the MODIFIED SIMPLER approach. Unfortunately the artificial compressibility preconditioner, that performed quite well on the depth averaged saddle point problem, is useless here. It requires the construction of an (incomplete) LU-factorization for the grad-div stabilized system $\mathbf{A}_{uv} + \omega \mathbf{G}_{uv}\mathbf{M}_1^T\mathbf{M}_2\mathbf{D}_{uv}$, which is a matrix with much more entries than $\mathbf{A}_{uv}$ itself. This is mainly caused by the fact that the matrix $\mathbf{M}_1^T\mathbf{M}_2$ has many nonzeros.

Fortunately the MODIFIED SIMPLER preconditioner - we introduced the SIMPLER method in Section 6.2.1 - for $\mathbf{K}_{uv\overline{p}}$ is relatively cheap to construct and it appears to be a good preconditioner. For a more extended description of SIMPLE(R) including eigenvalue analysis see [85] and [50]. The modification of SIMPLER that we use is the following: instead of the diagonal of $\mathbf{A}_{uv}$ we use a 2×2 block-diagonal $\mathbf{D}_{\mathbf{A}_{uv}}$. Because the unknowns u and v are clustered, this matrix $\mathbf{D}_{\mathbf{A}_{uv}}$ includes the Coriolis force, which dominates the momentum equations. This modification is crucial, since the standard SIMPLER preconditioner overlooks the important Coriolis force and therefore it fails to converge for Equation (7.13).

The application of MODIFIED SIMPLER can be reduced to the application of MODIFIED SIMPLE and SIMPLEL (see Section 6.3). Both require the solution of systems with $\mathbf{A}_{uv}$ and

$$\mathbf{C}_{SI} = \mathbf{M}_2\mathbf{D}_{uv}\mathbf{D}_{\mathbf{A}_{uv}}^{-1}\mathbf{G}_{uv}\mathbf{M}_1^T, \tag{7.16}$$

which is an approximation to the Schur complement. We will discuss the solution of these systems at the end of this section.

7.4.4 Solving the system for temperature and salt

Here we have to make a distinction between the two possibilities that we sketched in Section 7.3: (i) the BLOCK-GS approach with $\mathbf{S}_{TS} = \mathbf{A}_{TS}$ or (ii) the BLOCK-ILU approach with the Schur complement $\mathbf{S}_{TS}$ as in Equation (7.12).

The second approach is the most complex. It is expensive and in many cases even impossible to construct the exact Schur complement. To avoid the explicit construction we can use a Krylov subspace method (see Section 3.2) to solve the Schur complement. A Krylov method only requires that the matrix $\mathbf{S}_{TS}$ can be applied to a vector. This is relatively easy and cheap, because (7.12) shows that we can decompose it in a number of smaller matrix-vector products

and a few system solves (i.e. with $\mathbf{A}_p$, $\mathbf{A}_w$ and $\mathbf{K}_{uv\bar{p}}$, precisely the ones we just treated). The next question we have to answer is: how do we obtain a good preconditioner for $\mathbf{S}_{TS}$? It appears that in our case an incomplete LU factorization of $\mathbf{A}_{TS}$ suffices. One can get a better preconditioner by constructing a factorization for a better approximation to the Schur-complement. For example one can take into account more of the terms of (7.12). In general the construction of such a better approximation will be quite expensive and the number of nonzeros will increase rapidly, which makes the construction of an incomplete factorization much more expensive. We tried several approximations, but none of them did beat the relatively cheap incomplete LU factorization of $\mathbf{A}_{TS}$.

7.4.5 Nested iterations?

There are three systems left to solve: the large submatrices $\mathbf{A}_{TS}$ and $\mathbf{A}_{uv}$, and one of the two smaller systems $\mathbf{C}_{SI}$ (as in Equation (7.16)) or $\mathbf{A}_{\overline{uv}}^{GD}$ (as in Equation (7.15)). Unlike the systems $\mathbf{A}_p$ and $\mathbf{A}_w$ these four systems cannot be solved exactly at low cost. We use MRILU (see [10] and Section 3.2.2) to build a preconditioner for the matrices. For the matrices $\mathbf{A}_{TS}$ and $\mathbf{A}_{uv}$ we use clustering of the variables T,S and u,v respectively. MRILU is able to build a good factorization for these matrices.

We have to decide whether we use these preconditioners in an iterative method to solve the systems accurately or apply the preconditioner only once every time a subsystem needs to be solved. In the last case we in fact replace the matrices $\mathbf{S}_{TS}$ and $\mathbf{K}_{uv\bar{p}}$ in (7.11) with their preconditioners. We then need a relatively good MRILU factorization (i.e. with small drop tolerance). The number of outer iterations will increase anyway, but we avoid nested iteration schemes. If we apply nested iterations, we choose the flexible Krylov method (FGMRES, [62]) for the outer iteration.

7.5 Advantages of the tailored solver

At the end of this chapter we want to point at a few advantages of the BLOCK-GS/ILU solver over the previous solver: MRILU directly applied to system (7.1), which needed a huge amount of construction time, scaled badly and required much memory.

We expect the BLOCK-GS/ILU solver to scale almost linearly with the problem size. Most of the operations that have to be performed for the construction and application of the preconditioner, scale linearly with the problem size. For example, extraction of submatrices, computation of sparse matrix transposes, matrix-vector products and the solution of systems with lower or upper-triangular matrices can all be computed in $\mathcal{O}(d)$ time. The only trouble could come from the computation and application of the MRILU factorizations of the matrices $\mathbf{A}_{uv}$, $\mathbf{A}_{TS}$ and $\mathbf{A}_{\overline{uv}}^{GD}$ or $\mathbf{C}_{SI}$. Fortunately, these matrices are of convection-diffusion type and MRILU shows almost grid-independent convergence for that kind of problems [10].

We also expect that less memory is needed for the BLOCK-GS/ILU solver. MRILU applied directly to (2.34) needs much memory because a small drop tolerance is required. The BLOCK-GS/ILU preconditioner requires the storage of the subblocks and the MRILU preconditioners for the much smaller matrices $\mathbf{A}_{uv}$, $\mathbf{A}_{TS}$ and $\mathbf{A}_{\overline{uv}}^{GD}$ or $\mathbf{C}_{SI}$. Because MRILU is a suitable preconditioner for these matrices, the fill generated by the MRILU factorization will be rather small. Immediate consequence is that both construction and storage of the MRILU factors is much cheaper. Overall, we expect a serious decrease of the memory requirements when BLOCK-GS/ILU is used.

7.5.1 Data-assimilation

The solver that we developed in this chapter is primarily designed for use in the context of continuation of steady states, as we described in Section 2.3. However in the introduction in Chapter 1 we mentioned that we want to use the ocean model THCM in combination with the data-assimilation algorithm developed in [76]. In Section 2.6 we explained what this means for the solver: instead of systems with the steady state Jacobian Φ, we have to solve systems with the time dependent Jacobian Φ_t, as defined in Equation (2.43), as well as systems with its transpose Φ_t^T. Does the BLOCK-GS/ILU solver apply to these two systems?

Lets start with Φ_t. This matrix is given by $\Phi_t = 1/\Delta t\mathbf{M} + \Phi$, with $\mathbf{M}$ the diagonal matrix given in (2.42). The matrix has the same structure as (7.1), but the diagonals of $\mathbf{A}_{TS}$ and $\mathbf{A}_{uv}$ will be stronger, because we add $1/\Delta t$ to the diagonal entries. The solution of both subsystems plays an important role in the BLOCK-GS/ILU solver. As diagonal dominant matrices are easier to solve, we can expect the solution of time dependent systems to be much easier than the solution of steady-state Jacobian matrices, especially for small time steps Δt.

The solution of transposed system is no problem either. Except for the block $\mathbf{E}_{uv}$, the structure of the matrix in (7.1) is symmetric. It is precisely this submatrix that we ignored in the design of the BLOCK-GS/ILU preconditioner (see the first paragraph of Section 7.2). In the derivation of the preconditioner all properties of Φ that we used are properties of Φ^T as well. Hence the BLOCK-GS/ILU suits data-assimilation.

7.5.2 Parallelization

The last issue about the tailored solver that we want to address is parallelization. If the tailored solver indeed gives the desired speed up, we will increase the resolution and push the solver to the new limit. Which means that on a single processor PC we easily meet the memory limits again. Therefore we would like to be able to run the ocean model on a parallel super computer. Hence the question whether the BLOCK-GS/ILU solver can be parallelized is important.

Many operations in the construction and the application of the solver allow for parallelization. Extraction of submatrices, matrix-vector products, do not give any trouble here.

The solution of the triangular matrices $\mathbf{A}_w$ and $\mathbf{A}_p$ is no problem either, because with appropriate reordering these matrices are block-diagonal matrices with as many blocks as there are horizontal grid points. The blocks itself are smaller triangular matrices, they have dimension L, the number of grid points in the depth. As long as all nodes with the same θ and ϕ coordinate are stored on the same processor, no communication between processors is required to solve these systems.

Without doubt the bottle neck in the parallelization of the solver is the solution of the convection-diffusion problem for the tracers $\mathbf{A}_{TS}$ and the saddle point problem $\mathbf{K}_{uv}$. In Section 7.4.3 we exposed that the last problem can be reduced to the solution of $\mathbf{A}_{uv}$ and $\mathbf{A}_{\overline{uv}}^{GD}$ or $\mathbf{C}_{SI}$.

The parallelization of the solver has been carried out within a Trilinos [40] framework. Trilinos contains many parallel matrix solvers, like incomplete LU factorizations, multilevel methods and it can be coupled to the parallel direct matrix solver MUMPS [4]. As the parallel code of THCM is still under development, showing results would be premature. Nevertheless we can say that we succeeded to parallelize the construction and the application of the preconditioner and that scaling of both with problem size and number of processors is reasonable. However the tuning of the submatrix solvers takes some time before we can present results.

7.6 Conclusions

In this chapter we presented a new solver for the Jacobian matrices in THCM. The most important difference with the old solver is the fact that we exploit the structure of the matrix, which hides a cyclic dependency of the variables. To reveal this dependency, in Section 7.2 we transformed the pressure field and the continuity equations, swapped columns and rows and renamed certain subblocks. These operations resulted in the almost upper block-triangular system (7.9). If we simply drop the upper block (a coupling between temperature, salinity and pressure) we get a block-Gauss-Seidel solver (BLOCK-GS). If we leave it untouched, we can construct an exact block LU factorization. As it is unpractical to construct the full Schur complement involved in the exact factorization, it has to be solved iteratively, which gives the BLOCK-ILU solver. The application of either one of the two solvers requires the solution of a number of subsystems, which are all easier to solve than the system as a whole, because they are well known and understood.

There are a few open questions: which of the two (GS/ILU) gives the best preconditioner, can we profit from nested iterations and how is the performance compared to the old solver? We leave the answer to these questions for the next chapter where will use the solver in several large scale computations with THCM.

The advantages of the new solver are the expected decrease in memory requirements and construction time and better scalability. Hence the BLOCK-GS/ILU solver has the potential to give the desired speed up in large scale ocean circulation problem. In the next chapter we will see if it is able to do so in practice.

Chapter 8

Performance of the tailored solver in ocean flows

In Chapter 1 we promised to compute steady states of large scale ocean flows, nevertheless in the Chapters 4 to 7 we went deep into numerical analysis. There were some glimpses of ocean flows in Chapter 6. And in the previous chapter the ocean flows, at least the structure of the equations, played a prominent role. However so far we did not show any serious calculation with the implicit ocean model as introduced in Chapter 2. In this chapter we present real ocean calculations. We demonstrate the power of the BLOCK-GS/ILU preconditioner in the implicit ocean model in a pseudo-arclength continuation set-up.

The first problem is the double-gyre wind-driven circulation in a small basin (Section 8.1). The double-gyre flow consists of two cells rotating in opposite direction. The second problem is the computation of wind- and buoyancy driven single-hemispheric flows in an Atlantic size single-hemispheric basin (Section 8.2). The third problem is an abstraction of the global model: two double hemispheric basins connected through a periodic channel at the southern hemisphere.

These geometrically simplified models are of interest, because they give better insight in the effect of parameter changes, as the effects of the topography are minimized. Much of the phenomena that occur in more realistic topography are present in simpler models as well.

The final problem is the global ocean circulation (Section 8.4) with realistic topography. The aim of the presentation of the results is to demonstrate the capabilities of the implicit model when combined with the BLOCK-GS/ILU preconditioner and hence the physics of the solutions will not be discussed in detail.

The solutions of the problems in this chapter are three-dimensional, which we will depict in two-dimensional figures. Hence we use the *barotropic streamfunction* Ψ_B, which is the streamfunction of the vertically averaged velocity field, and the *meridional overturning streamfunction* Ψ_M, which is the streamfunction of the zonally averaged velocity field.

8.1 The double-gyre problem

As in the studies with quasi-geostrophic (QG) and shallow-water (SW) models in [17, Ch.5], a basin of 10° length and 10° width centered around 45°N is considered. In longitude ϕ and latitude θ, the boundaries of the domain are given by $\phi_w = 270^\circ$E, $\phi_e = 280^\circ$E, $\theta_s = 40^\circ N$ and $\theta_n = 50^\circ N$, so the size of the basin is approximately 1000×1000 km^2. The basin has a constant depth D = 2400 m. The flow is only forced by a steady wind-stress and otherwise has a constant temperature and salinity. A so-called double-gyre wind forcing is prescribed of the form

$$\tau^\phi(\phi,\theta) = -\tau_0 \cos\left(2\pi\frac{\theta-\theta_s}{\theta_n-\theta_s}\right), \tag{8.1a}$$

$$\tau^\theta(\phi,\theta) = 0 \tag{8.1b}$$

$$\tag{8.1c}$$

where τ_0 is a typical amplitude.

In the case of no thermal and freshwater forcing, $T_S = S_S = 0$, we know from QG and SW models [17] that multiple paths for the separation of the western boundary current may exist due to symmetry breaking of the wind-driven flow. As far as we know, we were the first to demonstrate this in primitive equation models [56]. In these equations there is no reflection symmetry with respect to the midaxis of the basin (here $\theta_0 = 45^\circ$) due to the spherical coordinates, and an imperfect pitchfork bifurcation is expected [17, Ch.5].

We fix the vertical resolution at $L = 12$, such that $H_m = 200$ m. To test the effects of the horizontal resolution on the pure wind-driven solutions of the model and the performance of the new block-ILU solver, we start with a value of $A_H = 1600$ m^{2}s^{-1}. For this value of A_H, a 40 km horizontal resolution (25 grid points) will still resolve the Munk layer at the western boundary, which has a thickness of

$$\delta_M = (A_H/\beta_0)^{1/3} \tag{8.2}$$

with $\beta_0 = 2\Omega\cos\theta_0/r_0$. So the solutions at three resolutions $25\times25\times12$, $50\times50\times12$ and $100\times100\times12$ can be compared.

We increase the wind stress parameter τ_0 from zero to the value of 0.1 Pa and then decrease A_H. Along the solution branch, the maximum value of the barotropic streamfunction is monitored and the result is plotted in Fig. 8.1. For $A_H = 1600$ m^{2}s^{-1} and $\tau_0 = 0.1$ Pa, there is a nice quadratic convergence in the maximum value of the barotropic streamfunction ψ_B, as shown in the second column of Table 8.1. The curves in Fig. 8.1 remain close to a value of $A_H \approx 650$ m^{2}s^{-1}. The curve for the $25\times25\times12$ grid starts to diverge first, since the resolution is not sufficient anymore to resolve the Munk boundary layer. The curve for the 20 km horizontal resolution (50 grid points) starts to diverge near $A_H = 400$ m^{2}s^{-1} for the same numerical reason. Based on the dependence of the Munk boundary layer thickness on A_H in Equation (8.2), the solutions for the $100\times100\times12$ grid are expected to be accurate for values of A_H down to $A_H \approx 250$ m^{2}s^{-1}. This is confirmed with the results of the $125\times125\times12$ grid.

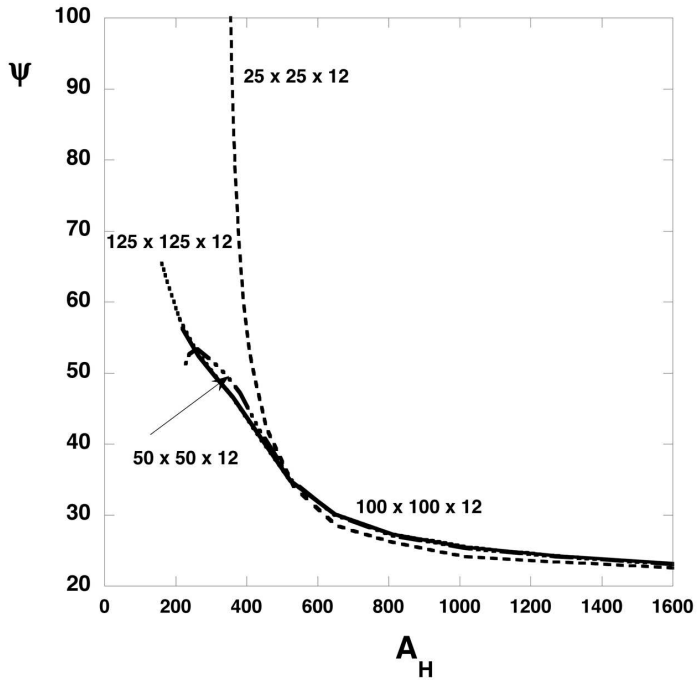


Figure 8.1: Solution branch of the constant density, wind-driven ocean circulation with control parameter A_H (in m^2s^{-1}) for the four different resolutions indicated. Ψ (in Sv, 1 Sv = 10^6 m^3s^{-1}) is the maximum of the barotropic streamfunction.

For each resolution, about 10-20 branch steps were taken to compute the curves in Fig. 8.1. To provide an impression of the performance of the BLOCK-ILU solver, that we described in Chapter 7, we provide in the other entries of Table 8.1 the average CPU time (in seconds on a 1GHz HP-DS25 workstation) of the building of the Jacobian matrix Φ, the construction of

GRID	ψ_B	d	MATRIX	BLOCK-ILU	#IT	FGMRES
$25\times25\times12$	22.53	45,000	0.9	1.8	7	0.8
$50\times50\times12$	23.01	180,000	3.8	9.1	9	5.5
$100\times100\times12$	23.14	720,000	16.3	48.5	11	42.9
$125\times125\times12$	23.15	1,125,000	24.9	84.7	8	48.7

Table 8.1: Overview of CPU time needed for the first iteration within the Newton-Raphson process with a step $ds = -10.0$ in A_H starting at $A_H = 1600$ m^2s^{-1}. Also the value of the maximum of the barotropic streamfunction ψ_B is shown for the four different resolutions at $A_H = 1600$ m^2s^{-1}. The quantity d is the number of degrees of freedom and the number of iterations (#it) in the FGMRES iterative process is also shown.

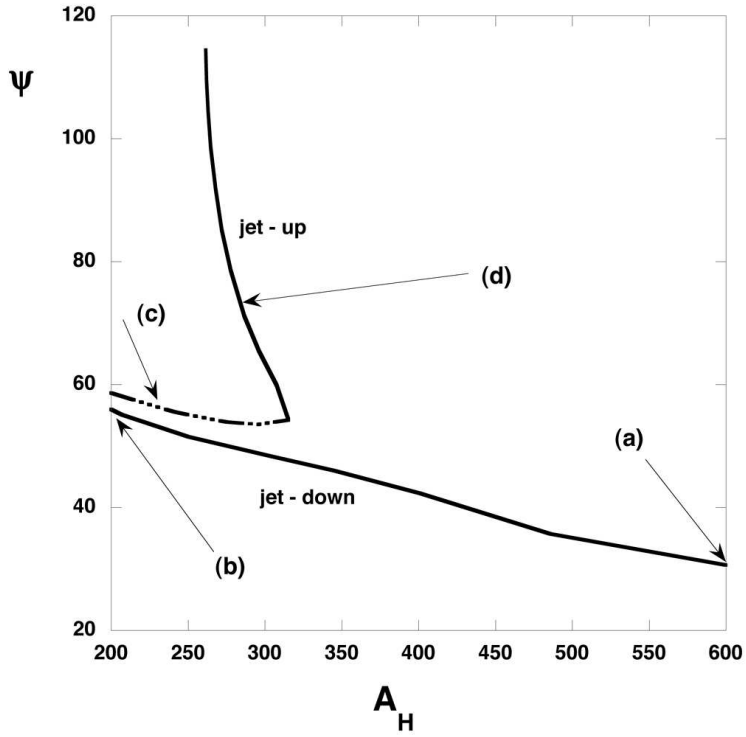


Figure 8.2: Bifurcation diagram showing the maximum of the barotropic streamfunction ψ_B (Sv) of the constant density circulation versus the control parameter A_H (in m^2s^{-1}). Solutions at the labeled locations are presented in Fig. 8.3.

the BLOCK-ILU preconditioner and the FGMRES solution of the first of the two linear systems in Equation (2.28). For this computation we used the nested iterations in the BLOCK-ILU solver. The accuracy of the inner iterations is 10^{-2} for the systems $\mathbf{S}_{TS}$ and $\mathbf{A}_{uv}$ and 10^{-6} for the depth-averaged saddle point problem (see Equation (7.14)). The accuracy of the depth averaged saddle point problem has to be relative small, because the computed depth-averaged pressure is used to solve the velocity field $\mathbf{x}_{uv}$. Hence the accuracy of the velocity field is largely influenced by the error in the pressure. The method is not very sensitive for the accuracy of the inner iterations for $\mathbf{S}_{TS}$ and $\mathbf{A}_{uv}$. A change in the accuracy will move effort from inner to outer iterations or vice versa, but the total computation time will be approximately the same.

As the temperature and salinity fields are constant, the matrices $\mathbf{B}_{uv}$ and $\mathbf{B}_w$, which contain derivatives of T and S, are zero matrices. Hence Equation (7.12) reduces to $\mathbf{S}_{TS} = \mathbf{A}_{TS}$ and we expect the BLOCK-ILU preconditioner to perform well. This is supported by the results. The number of outer iterations is small and the total CPU time per Newton step increases approximately linearly with the number of degrees of freedom.

In a typical bifurcation diagram, we use the lateral friction coefficient A_H as a parameter and a resolution of $100 \times 100 \times 12$ points (Fig. 8.2). In agreement to what is found in shallow-water models, we here find an imperfect pitchfork bifurcation, which leads to the existence of

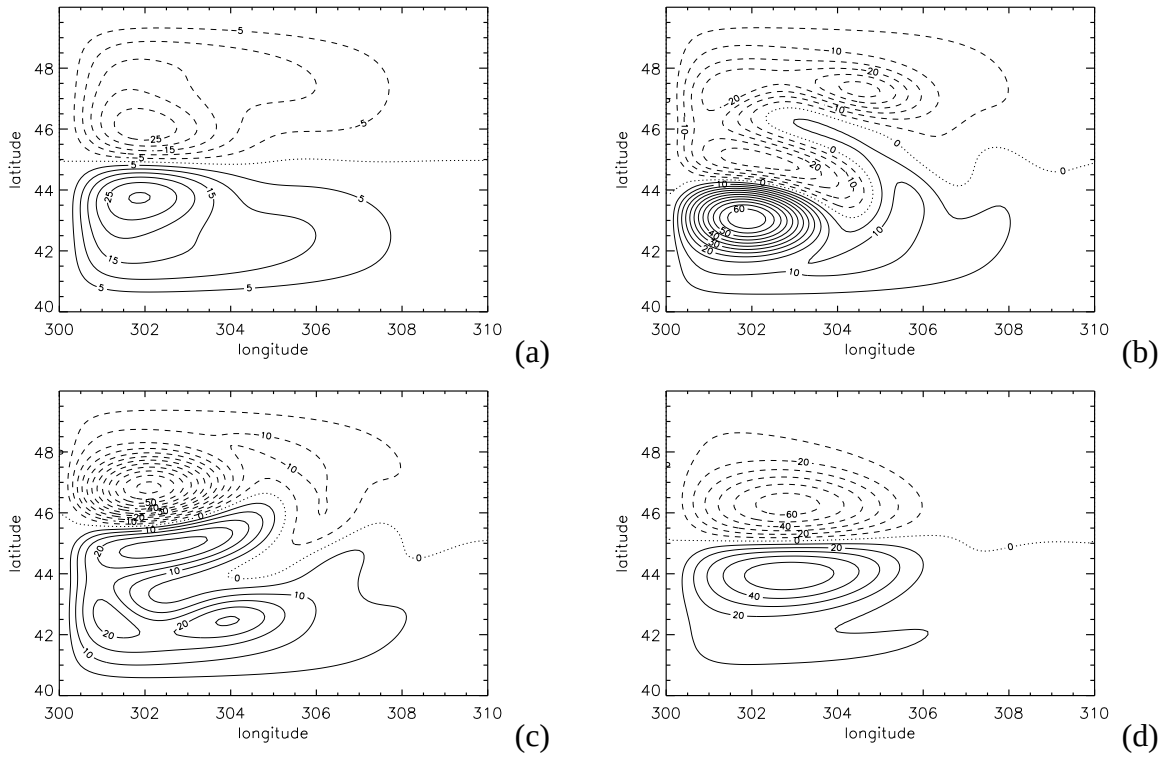


Figure 8.3: Patterns of the barotropic streamfunction of labeled locations in Fig. 8.2. Contour values are in Sv.

multiple patterns below a value of $A_H = 320$ m^{2}s^{-1}. The isolated branch in Fig. 8.2 was found by the residue continuation algorithm as described in chapter 4 of [17] and analyzed in [37].

The barotropic streamfunction of steady solutions at specific labeled locations in Fig. 8.2 are plotted in Fig. 8.3. The solution for large A_H on the connected branch is the near anti-symmetric double-gyre flow (Fig. 8.3a). When A_H decreases along this branch, a so-called jet-down solution appears (Fig. 8.3b), a solution very well known from QG and SW models [17]. Along the isolated branch, a jet-up solution exists (Fig. 8.3c) along the lower branch and after the saddle-node bifurcation at $A_H = 320$ m^{2}s^{-1}, the flow becomes more inertially controlled (Fig. 8.3d). Although these results were expected, it is the first time that such a multiple equilibria structure has been computed for a full primitive equation model.

8.2 Wind- and thermohaline flows

Similar to the study in [74], we consider a basin of 64° length and 64° width centered around 40°N. In longitude ϕ and latitude θ, the boundaries of the domain are given by $\phi_w = 286°$E, $\phi_e = 350°$E, $\theta_s = 10°N$ and $\theta_n = 74°N$; the basin has a constant depth D = 4000 m. As forcing

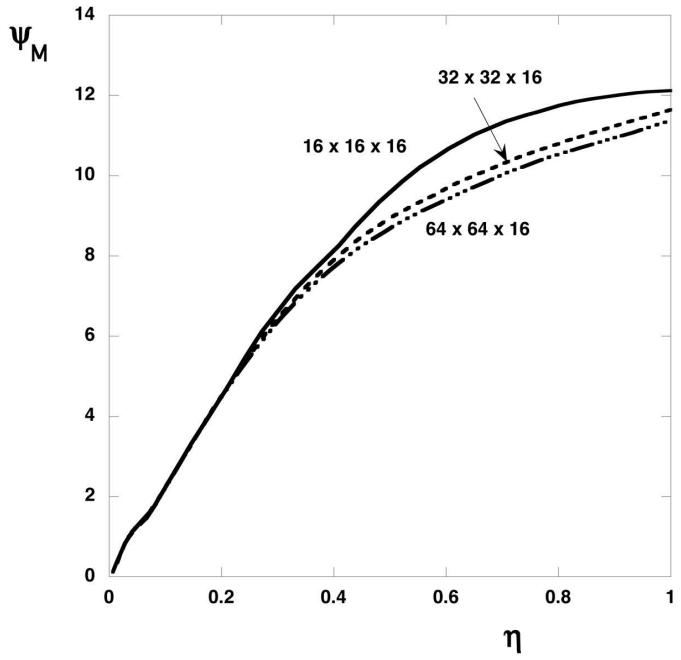


Figure 8.4: Solution branch, showing the maximum of the meridional overturning streamfunction ψ_M (in Sv) of the thermohaline and wind-driven ocean circulation versus the control parameter η for the three different resolutions indicated.

we choose

$$T_S(\phi,\theta) = T_0 + \eta\frac{\Delta T}{2}\cos\left(\pi\frac{\theta-\theta_s}{\theta_n-\theta_s}\right) \tag{8.3a}$$

$$S_S(\phi,\theta) = S_0 + \eta\frac{\Delta S}{2}\cos\left(\pi\frac{\theta-\theta_s}{\theta_n-\theta_s}\right) \tag{8.3b}$$

where we have introduced another homotopy parameter η such that $\eta = 0$ represents no forcing and $\eta = 1$ is a realistic strength of the forcing. Furthermore, we use interpolated values of the annual mean wind-stress dataset of [79] and multiply the amplitude of this forcing by η. In this way, we can vary η and increase all three forcing functions simultaneously.

We take a constant vertical mixing coefficient, i.e. $\eta_V = 0$ in (2.12) with $K_V^0 = 10^{-4}$ m^{2}s^{-1} and $K_V^c = 10^2$ m^{2}s^{-1}. In addition, we use values $A_H = 2.5\times10^5$ m^{2}s^{-1}, $K_H = 1.0\times10^3$ m^{2}s^{-1}, $\Delta T = 20^\circ$C and $\Delta S = 1.0$ psu (practical salinity unit). Furthermore, we do not yet consider neutral mixing ($\eta_M = 0$) and GM-mixing ($\eta_G = 0$). The vertical resolution is chosen as $L = 16$ such that $H_m = 250$ m. Maximum values of the meridional overturning streamfunction (ψ_M) are plotted versus η in Fig. 8.4 for three different resolutions; the highest resolution corresponds to a horizontal resolution of 1°. Beyond $\eta = 0.3$, the three curves start to deviate from each other and a smaller value of the meridional overturning occurs for the higher resolution model at larger values of η.

RESOLUTION	ψ_M	d	MATRIX	BLOCK-ILU (MRILU)	#IT	FGMRES (GMRES)
$16\times16\times16$	12.12	24,576	2.0	9.1 (260.6)	37	44.2 (81.1)
$32\times32\times16$	11.64	98,304	8.6	70.6	60	330.2
$64\times64\times16$	11.36	393,216	36.9	557.2	35	882.9

Table 8.2: Overview of CPU time needed for the first iteration in the Newton-Raphson process for a step $\Delta s = 0.01$ in η starting at $\eta = 1$. The format is the same as in Table 8.1. For the lowest resolution, also timing results for the original MRILU solver are provided between brackets. The maximum of the meridional overturning streamfunction ψ_M for three different resolutions and $\eta = 1$ is also presented.

The maximum of meridional overturning streamfunction (ψ_M) converges with the horizontal resolution as can be seen from the second entry of Table 8.2. From each solution at $\eta = 1$, one step with $\Delta s = 0.01$ was performed in η and the CPU time required for the first iteration in the Newton-Raphson process is shown in Table 8.2. It is clear that computations involving these thermohaline and wind-driven flows are more expensive than those for only wind-driven flows, since the flow is simply more complex with salt and temperature variations over the computational domain.

In column 4 one observes that the time for constructing the matrix is linear in the number of unknowns as may be expected. However this is not the case for the construction of the preconditioner (in column 5). A similar behavior is shown in [19] for MRILU applied to the whole system at once, and will also occur if a direct method with fill-reducing order is used. However on the coarsest grid the construction time of the preconditioner in the new solver is more than an order of magnitude less than that in the original MRILU approach. Even if the construction time growths faster than linear with the problem size, the new approach is a large improvement. Moreover the behavior results from the construction of the MRILU preconditioner for A_{ST}, and is due to the dropping parameters used. These are currently not optimized. We see that the number of iterations in the 6^{th} column has an outlier (60 is much larger than 37 and 35), which also makes the solution phase relatively expensive here. Correcting for this number of iterations, we observe a solution time (last column) which increases approximately linear with the number of unknowns.

For the lowest resolution, a comparison with MRILU applied to the whole system is provided in the 5^{th} and 7^{th} column of Table 8.2. The timing results of MRILU are between brackets. The new solver is a significant improvement over the old approach, in particular the construction of the preconditioner is more than a magnitude faster needing also less computing time for the solution phase. Due to lack of memory we were not able to apply MRILU for the finer resolutions.

Patterns of the meridional overturning streamfunction ψ_M and the barotropic streamfunction ψ_B are plotted in Fig. 8.5 at $\eta = 1$ for each of the three resolutions. Each solution is completely stably stratified and sinking occurs north of 60°N. In principle, the solutions for the coarsest grid are already sufficiently accurate for this value of K_V^0 and A_H and they are just smoothed at higher resolution.

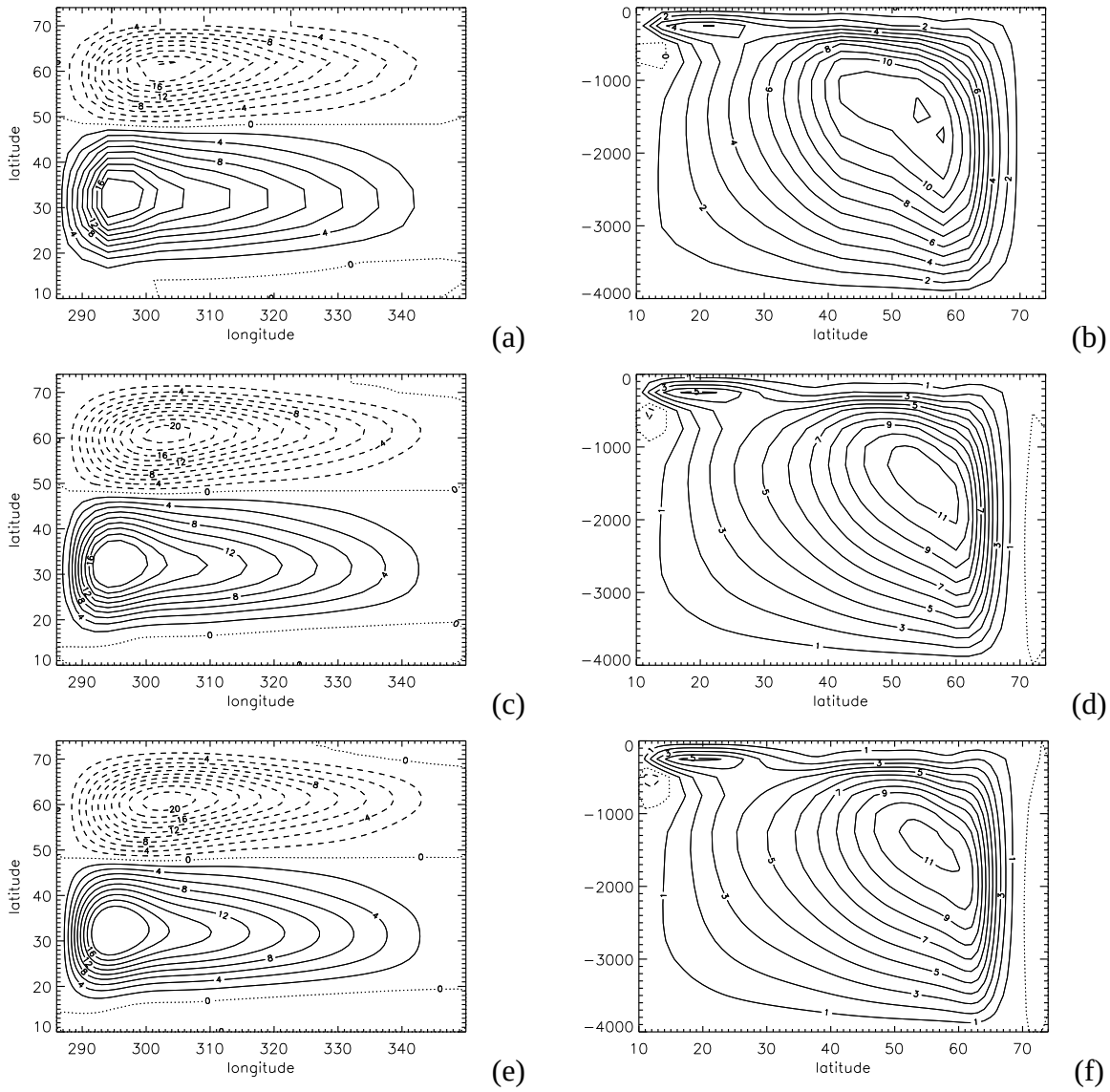


Figure 8.5: Patterns of the barotropic streamfunction (a, c, e) and meridional overturning streamfunction (b, d, f) at $\eta = 1$. (a-b): $16 \times 16 \times 16$, (c-d): $32 \times 32 \times 16$ and (e-f): $64 \times 64 \times 16$.

8.3 Double basin

The third setup is the double basin problem, which can be viewed as an abstraction of the global ocean model. The problem is similar to the one in [41, 18]. We have two basins of equal size; the basins are connected by a channel with periodic boundary conditions at the southern hemisphere. The boundaries of the domain in longitude ϕ and latitude θ are given by $\phi_w = 120°\text{E}$, $\phi_e = 255°\text{E}$, $\theta_s = -68°N$ and $\theta_n = 68°N$; the basin has a constant depth D = 4,500 m.

In the channel there are two sills, one of 2,100 m in the middle below the continent and one of 1,200 m at the periodic boundaries. We use two grids, one of $36 \times 34 \times 15$ (4° horizontal resolution) and one of $72 \times 68 \times 15$ (2°).

As forcing for the temperature and salinity we choose the functions

$$\begin{aligned} T_S(\phi,\theta) &= T_0 + \eta \frac{\Delta T}{2} \cos\left(\pi \frac{\theta}{\theta_n}\right), \\ S_S(\phi,\theta) &= S_0 + \eta \frac{\Delta S}{2} \cos\left(\pi \frac{\theta}{\theta_n}\right), \end{aligned}$$

with $\Delta T = 20°\text{C}$ and $\Delta S = 1.0$ psu.

For the wind forcing we use the function

$$\begin{aligned} \tau^{\phi}(\phi,\theta) &= \eta\,\tau_0 \left(0.2 - 0.8\sin(6\theta) - 0.5\left(1 - \tanh(10\theta)\right)\right. \\ &\quad \left. -0.5\left(1 - \tanh\left[10\left(\pi/2 - \theta\right)\right]\right)\right), \\ \tau^{\theta}(\phi,\theta) &= 0, \end{aligned}$$

where $\tau_0 = 0.1$ Pa is a typical amplitude. Finally we choose $A_H = 2.8 \cdot 10^5$ m^{2}s^{-1} and the other parameters as in Table 2.1.

For both resolutions we continued the forcing parameter η to 1. Both runs were performed on a PC with two 2.4GHz AMD Opteron processors and 7.6GB memory. The runs of the 4° and 2° resolution took approximately 9 and 15 hours respectively. The increase in time is less than a factor 2, whereas the number of unknowns grows with a factor 4. This mild increase in time is caused by the fact that the continuation process on the coarsest grid had difficulties following the branch for increased forcing. Near $\eta = 0.95$ the step size was halved several times, which meant we needed more steps to reach the target value 1.0. Note that this is not a problem of the solver, which converged smoothly, but of the Predictor-Corrector iteration in the continuation method (see Section 2.3.2). On the finest grid the continuation method could take the maximum step size $\Delta s = 0.1$ psu. Probably this is caused by the fact that the physics is better represented on the finest grid.

In Figure 8.6 we show the streamfunction and the meridional overturning in the Pacific and the Atlantic basin. The patterns correspond to the ones found in [18].

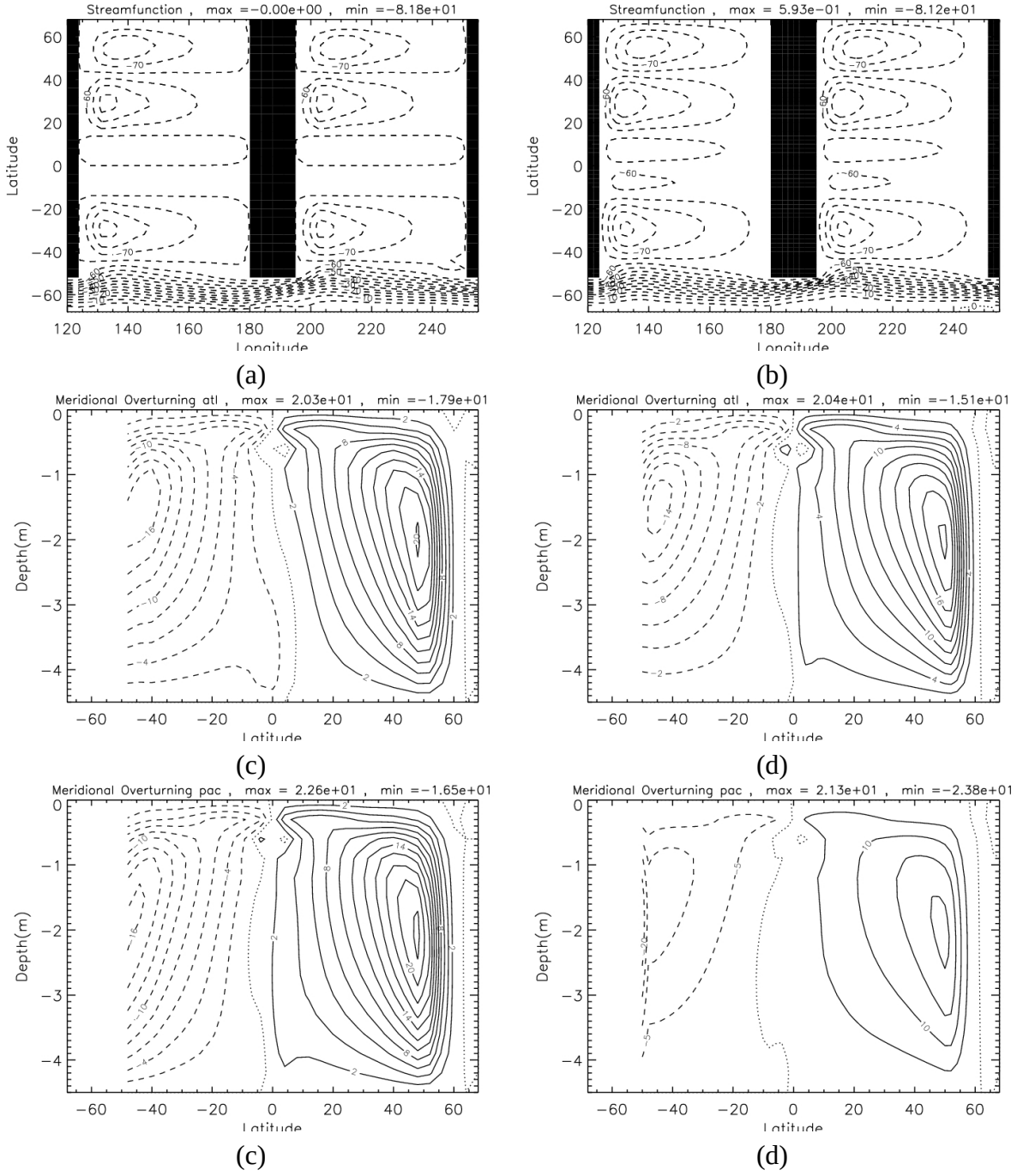


Figure 8.6: Patterns of the barotropic streamfunction Ψ_B (in Sv) (a-b) and the meridional overturning streamfunctions Ψ_M (in Sv) in the Atlantic (c-d) and the Pacific (e-f) at $\eta = 1$. The maximum and minimum value of the streamfunction can be found just above the figures. (a,c,e): $38 \times 34 \times 15$, (b,d,f): $76 \times 68 \times 15$.

RESOLUTION	d	MATRIX	TYPE	PREC	(F)GMRES	#it
$36 \times 34 \times 15$	110,160	9.4	BLOCK-GS	34.7	74.8	265
			BLOCK-ILU	34.8	521.5	74
			MRILU	14,950.4	285.5	$200^{\dagger}$
$72 \times 68 \times 15$	440,640	37.6	BLOCK-GS	115.3	488.6	$300^{\dagger}$
			BLOCK-ILU	117.5	3,717.5	$99^{\dagger}$

Table 8.3: Overview of CPU time needed for the first iteration in the Newton-Raphson process for a step $\Delta s = 0.01$ in η starting at $\eta = 1$. The column d contains the number of unknowns, the column MATRIX shows the CPU-time needed for the construction of the Jacobian. The column TYPE gives the type of preconditioner. The last three columns contain the CPU-time for the construction of the preconditioner, the solution of the system with the Jacobian and the number of iterations in (F)GMRES. A † means that (F)GMRES reached the maximum number of iterations before the desired accuracy (10^{-6}) was obtained.

In Table 8.3 we show the CPU-times for a single solve of the Jacobian matrix with the preconditioner that we presented in the previous chapter. We tested both variants of the preconditioner: block GS and block ILU (see Section 7.4.5). The BLOCK-GS preconditioner uses the modified SIMPLER method as preconditioner for saddle point problem (7.13). Moreover the systems $\mathbf{K}_{uv}$ and $\mathbf{A}_{TS}$ are solved with only one application of the corresponding preconditioners.

The BLOCK-ILU method uses nested iterations to solve $\mathbf{K}_{uv}$ and the real Schur complement $\mathbf{S}_{TS}$ (see Equation (7.12)). The saddle point problem is solved with an artificial compressibility preconditioner for the depth averaged saddle point problem in Equation (7.14). We compare both preconditioners to MRILU directly applied to the Jacobian matrix.

First of all we see that MRILU requires an enormous amount of CPU-time to construct a preconditioner. The construction takes more than 4 hours, which makes it 400 times as expensive as the construction of the BLOCK-GS/ILU preconditioner. The large amount of construction time might be caused by the fact that MRILU runs out the 7.8GB memory and uses virtual (disc) memory which requires expensive memory-swapping. The other two preconditioners don't suffer this lack of memory, they require approximately 1.5GB.

There is more bad news for the MRILU preconditioner: the Krylov method converges very slow. The maximum number of iterations is reached without any sight of convergence to the desired accuracy (10^{-6}). The accuracy in the last iteration was only 10^{-2}. Both BLOCK-GS and BLOCK-ILU converge within the maximum number of iterations, which is 300 and 99 respectively. For BLOCK-ILU we use a smaller maximum, because the nested Krylov methods make each iteration more expensive. The CPU-times show that we better use the straight forward block Gauss-Seidel method. The effort in the inner iterations does speed up the convergence in the outer iterations, but not enough to compensate the extra cost.

In Table 8.3 we also show results for the BLOCK-GS/ILU preconditioners on the finest grid. Note that the construction time for the matrix and the preconditioner scale linear with the problem size. The solution time scales approximately with a factor 7. Both Krylov methods stop when the maximum number of iterations was reached, however the accuracy in the last

iterations was close to the desired 10^{-6}. Convergence would have been obtained within 350 iterations.

Based on the results for the double basin case, we can conclude that the BLOCK-GS preconditioner gives an enormous speed up of the computations.

8.4 Global thermohaline circulation

The last setup is a more realistic global ocean circulation problem from [86]. We have $\phi_e = -180^\circ$E, $\phi_w = 180^\circ$E, $\theta_s = -85.5^\circ$N and $\theta_n = 85.5^\circ$N. Moreover we use continents, bottom topography (deepest point at 5,000 m) and a vertically refined grid near the surface. The layer at the top has a thickness of 50 m and the layer at the bottom 1,000 m. The bathymetry is based on the ETOPO-10 data set, which is interpolated onto the model grid and smoothed. The grid has $96 \times 38 \times 12$ cells, which gives a 4° horizontal resolution. We use the standard parameter values, as given in Table 2.1, except for A_H that gets the value $2.5 \cdot 10^5$ m^{2}s^{-1}.

For the wind forcing we use interpolated values of the annual mean wind-stress dataset of [79]. For the salinity and temperature forcing we use the data sets of [48, 49]. We multiply the amplitude of all three forcings with η. In this way, we can vary η and increase all three forcing functions simultaneously.

Similar to the previous setups we continued the forcing parameter η to 0.96. We couldn't reach 1.0 due to convergence problems of the Predictor-Corrector method (see Section 2.3.2), which might be caused by the bottom topography and the course grid. We observed that large residues occurred first near areas of recirculation, which might explain the problems in the convergence. We remark that the solver has no problems in solving the equations with the Jacobian.

The computations were done on a PC with two 2.4GHz AMD Opteron processors and 7.6GB memory. Due to the complex geometry, we had to use a small step size, so $\Delta s \leq 0.01$, which caused the total run to take approximately 4 days of computation time.

The barotropic streamfunction and the temperature and salinity profiles in the top layer (just below the surface) can be found in Figure 8.8. The meridional overturning in Atlantic and Pacific are shown in Figure 8.7. The overturning is in the right order of magnitude.

The results for the global ocean circulation in [86] were obtained with an earlier version of THCM that used a C-grid, no convective adjustment and a much higher value of the horizontal viscosity $A_H = 1.6 \cdot 10^7$. Comparing our results to the ones in that paper, we see that the Antarctic Circumpolar Current (ACC) in Figure 8.8 (top) is much stronger. The maximum of the barotropic streamfunction in the ACC is approximately 100 Sv, which is in the same order of magnitude as the observed strength of 135 Sv. The better results are due to the smaller value of A_H, which is possible because we use a B- instead of a C-grid (see the discussion in Section 2.2.3).

The extrema in the Meridional Overturning in the Atlantic and the Pacific as depicted in

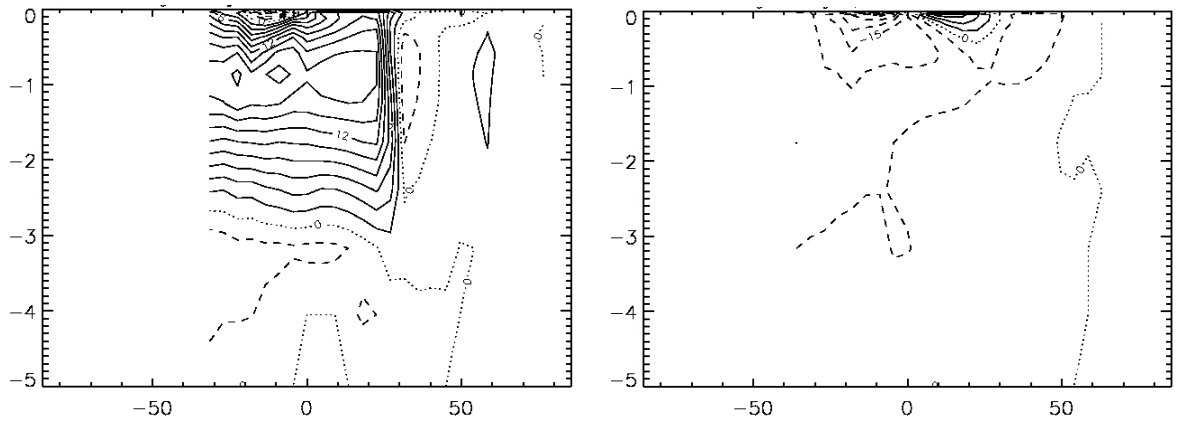


Figure 8.7: Pattern of the meridional overturning streamfunction Ψ (in Sv) in the Atlantic (left), with $\Psi_{\min} = -0.8$ Sv and $\Psi_{\max} = 18.9$ Sv, and the Pacific (right), with $\Psi_{\min} = -31.6$ Sv and $\Psi_{\max} = 17.7$ Sv, for $\eta = 0.96$.

Figure 8.7 agree with observations as well. The sinking in the North Atlantic is at 30°N, while it should be at 50°N. This could be explained by the fact that we use the Levitus forcing for the temperature in the top layer. In Figure 8.8 (middle) one clearly sees that the surface temperature in the North Atlantic drops down quickly near 30°N. A relaxation of the temperature forcing at the surface by the introduction of an atmospheric layer, like we did in [86], would possibly move the sinking to the north.

8.5 Summary and discussion

In this chapter we showed four typical examples of ocean flows to demonstrate the performance of the tailored linear system solver that we described in Chapter 7.

We are able to compute the bifurcation structure of the isothermal wind-driven double-gyre flow using a 10 km resolution involving the analysis of a system of 720,000 degrees of freedom. Encouraging result is here that the computational cost increases approximately linearly with the number of degrees of freedom. Furthermore, we are able to compute steady wind- and buoyancy driven flows in a single hemispheric Atlantic size basin with up to 1° horizontal resolution. The latter result opens the way to study the interaction of the thermohaline and wind-driven ocean circulation in more detail using tools of dynamical systems theory.

For the second and the third example we compared the timing results of the new solver to the results of the old solver, i.e. MRILU directly applied to the system. The tailored solver has several advantages: the construction of the preconditioner is much faster, the storage of the preconditioner requires less memory and the application of the preconditioner is much cheaper. The reduction of total solve time (preconditioner + FGMRES) lies somewhere between 90% and 99%. Furthermore it seems that the new solver scales well (close to linear) with the

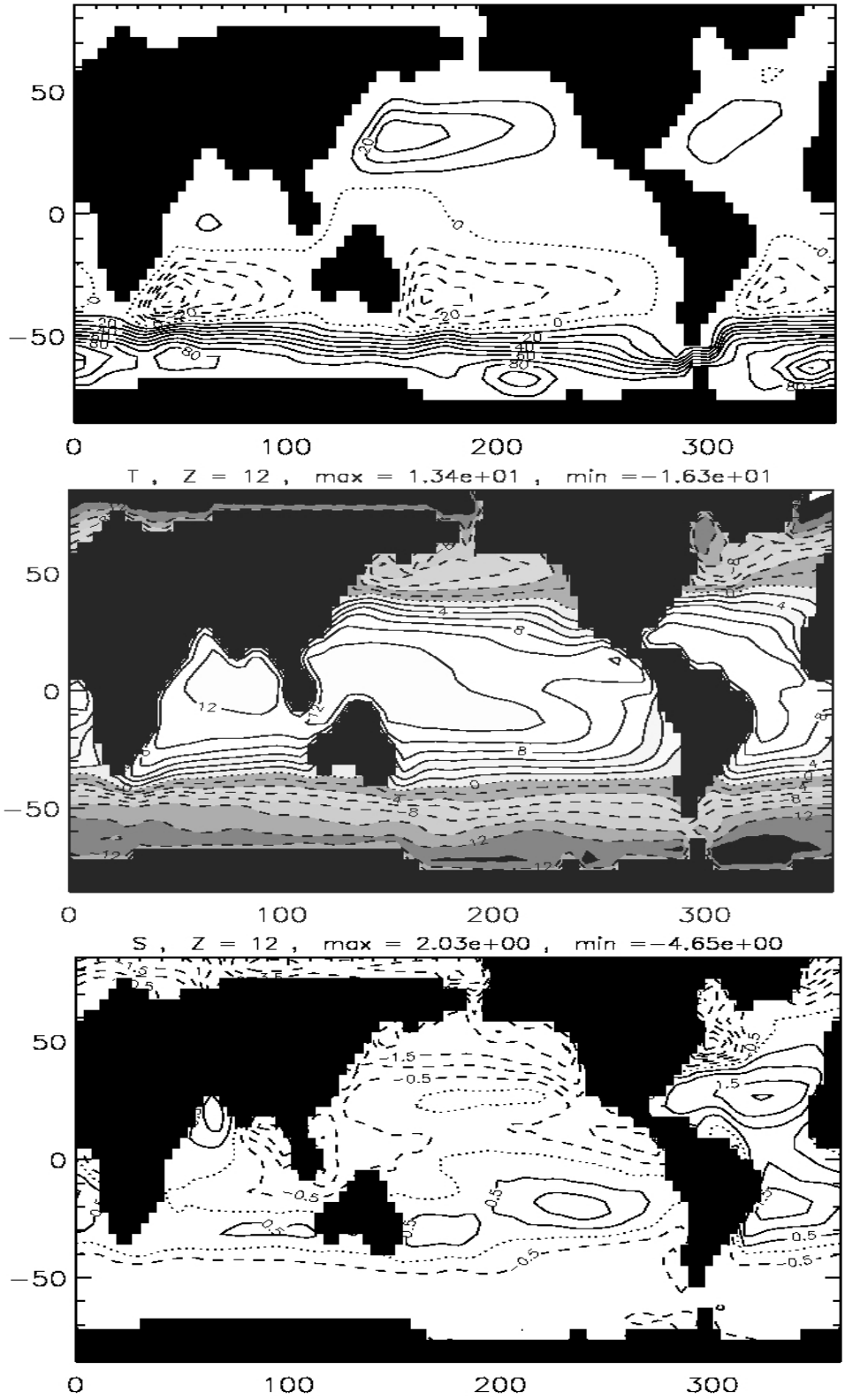


Figure 8.8: Pattern of the barotropic streamfunction (top) in Sv (max 100, min -40), the temperature profile in the top layer (middle) and the salinity profile in the top layer (bottom), all for $\eta = 0.96$.

problem size. We can conclude that the BLOCK-GS/ILU solver that exploits the structure of the equations, is a major step forward to apply bifurcation analysis to ocean models with up to $\mathscr{O}(10^6)$ degrees of freedom.

Chapter 9

Discussion

In the very first chapter we set the challenge for this thesis: *to develop new numerical techniques, such that it is possible to do large scale computations on steady states of the global ocean circulation with sufficient accuracy.*

In fact the improvement of the numerics was only the third of three goals of a larger research project. The major aim is the reconstruction of the time-mean absolute velocity field of the global ocean circulation. The way to go was to enhance the existing thermohaline ocean model THCM, which lead to three subgoals:

(1) improve the physics in the model,
(2) tune the model using observations,
(3) speed-up the numerics.

The research on the first two subgoals was carried out by Terwisscha van Scheltinga [75], who incorporated the advanced mixing schemes of Section 2.2.2 in THCM, and developed an implicit data-assimilation algorithm, called I4DVAR. We described data-assimilation and the corresponding extra conditions on the solver in Section 2.6. We focussed on (3) the numerics. The bottleneck in the computations of THCM at the start of the project was the solution of large linear systems involving the Jacobian matrix. Hence the plan to reach goal (3) was to develop a new solver, that

(3a) gives a considerable speed up of the computations,
(3b) allows for parallelization,
(3c) combines with data-assimilation.

In Chapter 2 we described the ingredients of THCM: the (discrete) equations, numerical continuation of steady states and the structure of the Jacobian matrix. However in a large part of the thesis we focussed on the somewhat easier saddle point problem (see Section 2.5). We did so, because iterative solvers have a hard time in solving the saddle point problem, probably for the same reason they have difficulties with the ocean matrix. In the preceding chapters of this thesis we investigated several methods to solve either the saddle point or the ocean problem and in the present chapter we will summarize the results, draw conclusions and evaluate to what extend we reached our goals.

Direct and iterative methods are hard to combine for the saddle point problem

We started off searching the new solver in a combination of techniques from iterative and direct methods, which we described in Chapter 3. Hence we studied the direct solution of saddle point matrices in Chapter 4. We proposed a new algorithm that computes an ordering for the matrix, that is feasible, gives sparse factors and it is numerically stable. One of the nice things of the algorithm is that it keeps much of structure of the saddle point problem during the factorization phase. Moreover we can prove that the size of the matrix elements encountered during factorization is bounded linearly by the size of the matrix.

We tried to transform this exact factorization into an incomplete factorization, but all attempts failed. The main problem is that the factorization is very sensitive to dropping in the divergence part of the saddle point problem. The number of iterations in the Krylov method grows rapidly even if we drop only a few small elements.

Similar problems were encountered in Chapter 5, where we combined an incomplete LU factorization with the fill-reducing orderings as used in direct methods. We introduced a new type of equivalent matrix ordering: the bare-branch ordering. The ordering can be used in an ordered incomplete LU factorization algorithm, which uses both exact and approximate elimination. The performance of the algorithm is good for the equations that involve only one type of variable, like the Poisson equation. However the results for the saddle point problem are disappointing: the factorization has either a huge amount of fill or gives bad convergence in the Krylov space.

We conclude that, notwithstanding the remark in [65] that the combination of techniques from direct and iterative methods is the way to go, it is hard to successfully combine both approaches. This holds at least for the saddle point problem, but probably for a wider class of partial differential equations that involve different types of variables and constraints.

Coupled partial differential equations need solvers that respect the structure of the equations

In Chapter 6 we showed that for different types of saddle point problems, solvers that respect and even exploit the differences between the involved variables, perform well. We proposed two new preconditioners for the saddle point matrix: one based on grad-div adding, the other on artificial compressibility. Both preconditioners give the velocity and the pressure nodes a different treatment. We compared the preconditioners to other well-known saddle point preconditioners that take benefit from the structure of the matrix. Eigenvalue analysis and numerical experiments showed that our preconditioners are good alternatives for existing methods. Nevertheless the solution of the involved grad-div added matrix ($A + \omega B^T B$) requires some attention.

The positive results of approaches that exploit the structure on the saddle point problem, motivated to use such an approach in the design of a new solver for the Jacobian matrices that occur in THCM. In Chapter 7 we derived the new solver. It starts with a transformation (related to depth-integration) of the pressure and the continuity equation. The transformed matrix is block-lower triangular except for one upper diagonal block, which represents the presence of temperature and salinity in the hydrostatic pressure equation. Either we ignore this block,

which gives the BLOCK-GS preconditioner, or we leave it untouched, which gives the BLOCK-ILU preconditioner. The latter uses an approximation of the exact Schur complement.

Both preconditioners lead to successive solves of an equation for the pressure, a saddle point problem for the horizontal velocity and the depth-averaged pressure, an equation for the vertical velocity and finally a convection-diffusion problem for the tracers temperature and salinity. These subsystems are all quite well known and relatively easy to solve.

In Chapter 8 the BLOCK-GS/ILU solver is used in large scale ocean flow problems. The results show that this new solver considerably reduces the memory requirements and the computation time.

Based on the experience with both saddle point problems and the ocean matrices, we conclude that complex coupled partial differential equations require tailored solvers that respect and benefit from the structure of the equations.

Explicit methods can inspire the solvers for implicit methods

The BLOCK-GS solver shows a striking similarity to the barotropic-baroclinic splitting that is widely used in explicit ocean models. The same holds for one of the saddle point preconditioners that we introduced in Chapter 6. Artificial compressibility is used in explicit time step methods for incompressible flow problems, however it appears to be useful in the preconditioner as well.

The explanation for this phenomenon might be that the methods used in explicit time stepping resemble the dependencies in the solution of the problem: the most important variables are solved first. In implicit time stepping the time steps are much larger (even infinite in a steady state solution), but the dependencies are probably largely the same, which can be used in the design of a solver for the involved Jacobian matrices.

The new solver is a major step forward in bifurcation analysis of ocean flows

The main goal of this thesis was the development of a new, faster solver for the ocean systems. We already mentioned the design of the preconditioner in Chapter 7. The results of the preconditioner are above expectations. We compared the performance on four different large scale complex ocean flow problems to the performance of the old solver: MRILU directly applied to the Jacobian matrix. We observed speed-up factors of two orders of magnitude. Moreover the BLOCK-GS/ILU preconditioner uses much less memory and the scaling results are acceptable even for complex evolved flows. In a direct comparison of the two variants, i.e. the straightforward BLOCK-GS and the more complex BLOCK-ILU solver (nested iterations, approximation of the Schur-complement), we see that BLOCK-GS is the better one. It uses more outer iterations, but each one of the iterations is cheaper than an iteration with BLOCK-ILU.

There is no doubt we fulfilled condition (3a): the speed-up is considerably large. The parallelization (condition (3b)) follows naturally, as we exposed in Section 7.5.2. The parallel code is still under development. The combination with the data-assimilation code (3c) appeared

to be no problem either. In contrary, due to a stronger diagonal, the performance of the BLOCK-GS/ILU solver is even better than on steady state Jacobian matrices [75].

All together we conclude that the new solver allows us to perform computations with a much higher resolution, which makes it a major step forward in the analysis of the thermohaline ocean circulation.

Methodology ready for reconstruction of the time-mean absolute velocity field of the global ocean circulation

Of the three main goals, in this thesis we realized (3) a fast solver for the ocean equations. The other two, i.e. (1) the improvement of the physics and (2) the use of observations in the model, were realized by Terwisscha van Scheltinga at the IMAU [75]. Together we delivered an up-to-date ocean model THCM that is able to do the necessary large scale computations on the thermohaline circulation. The model can be combined with data-assimilation in I4DVAR using observations of the geoid an the sea surface height. Accurate data for the geoid will be provided by the GOCE mission. Unfortunately the launch of the satellite has been delayed to the end of 2007. Hence we have to wait for the data, but the methodology is ready for the reconstruction of the time-mean absolute velocity field of the global ocean circulation.

Bibliography

[1] E. L. Allgower and K. Georg. *Numerical Continuation Methods, An Introduction*. Springer-Verlag, Berlin, 1990.

[2] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.*, 17(4):886–905, 1996.

[3] P. R. Amestoy, T. A. Davis, and I. S. Duff. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Software*, 30(3):381–388, 2004.

[4] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Comput.*, 32(2):136–156, 2006.

[5] O. Axelsson. Preconditioning of indefinite problems by regularization. *SIAM J. Numer. Anal.*, 16(1):58–69, 1979.

[6] M. Benzi, G. H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numer.*, 14:1–137, 2005.

[7] M. Benzi and M. A. Olshanskii. An augmented Lagrangian-based approach to the Oseen problem. *SIAM J. Sci. Comput.*, 28(6):2095–2113 (electronic), 2006.

[8] Z. Bohte. Bounds for rounding errors in the Gaussian elimination for band systems. *J. Inst. Math. Appl.*, 16(2):133–142, 1975.

[9] E. F. F. Botta, K. Dekker, Y. Notay, A. van der Ploeg, C. Vuik, F. W. Wubs, and P. M. de Zeeuw. How fast the Laplace equation was solved in 1995. *Appl. Numer. Math.*, 24(4):439–455, 1997.

[10] E. F. F. Botta and F. W. Wubs. Matrix renumbering ILU: an effective algebraic multilevel ILU preconditioner for sparse matrices. *SIAM J. Matrix Anal. Appl.*, 20(4):1007–1026 (electronic), 1999.

[11] C. Brezinski. *Projection Methods for Systems of Equations*. North-Holland Publishing Co., Amsterdam, 1997.

[12] J. R. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Math. Comp.*, 31(137):163–179, 1977.

[13] J. R. Bunch and B. N. Parlett. Direct methods for solving symmetric indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 8:639–655, 1971.

[14] T. Coleman, B. S. Garbow, and J. J. Moré. Software for estimating sparse Jacobian matrices. *ACM Trans. Math. Software*, 1984.

[15] G. Danabasoglu and J. C. McWilliams. Sensitivity of the global ocean circulation to parameterizations of mesoscale tracer transports. *J. Phys. Oceanogr.*, 8:2967–2987, 1995.

[16] E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang. Towards a cost-effective ILU preconditioner with high level fill. *BIT*, 32(3):442–463, 1992.

[17] H. A. Dijkstra. *Nonlinear Physical Oceanography: A Dynamical Systems Approach to the Large Scale Ocean Circulation and El Niño*, volume 28 of *Atmospheric and Oceanographic Sciences Library*. Springer, New York, 2nd edition, 2005.

[18] H. A. Dijkstra and A. von der Heydt. Localization of multidecadal variability: II. Spectral origin of multidecadal modes. *J. Phys. Oceanogr.*, 2007. In press.

[19] H. A. Dijkstra, H. Öksüzŏglu, F. W. Wubs, and E. F. F. Botta. A fully implicit model of the three-dimensional thermohaline ocean circulation. *J. Comput. Phys.*, 173:685–715, 2001.

[20] C. R. Dohrmann and R. B. Lehoucq. A primal-based penalty preconditioner for elliptic saddle point systems. *SIAM J. Numer. Anal.*, 44(1):270–282 (electronic), 2006.

[21] M. R. Drinkwater, R. Floberghagen, R. Haagmans, D. Muzi, and Popescu A. GOCE: ESA's first earth explorer core mission. In G. B. Beutler, M. Drinkwater, R. Rummel, and R. von Steiger, editors, *Earth Gravity Field from Space - from Sensors to Earth Sciences*, volume 18 of *Space Sciences Series of ISSI*, pages 419–432. Kluwer Academic Publishers, Dordrecht, Netherlands, 2003. See also http:://www.esa.int/esaLP/LPgoce.html.

[22] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Monographs on Numerical Analysis. The Clarendon Press Oxford University Press, New York, 2nd edition, 1989. Oxford Science Publications.

[23] I. S. Duff, N. I. M. Gould, J. K. Reid, J. A. Scott, and K. Turner. The factorization of sparse symmetric indefinite matrices. *IMA J. Numer. Anal.*, 11(2):181–204, 1991.

[24] I. S. Duff and G. A. Meurant. The effect of ordering on preconditioned conjugate gradients. *BIT*, 29(4):635–657, 1989.

[25] I. S. Duff, J. K. Reid, N. Munksgaard, and H. B. Nielsen. Direct solution of sets of linear equations whose matrix is sparse, symmetric and indefinite. *J. Inst. Math. Appl.*, 23(2):235–250, 1979.

[26] H. C. Elman and D. J. Silvester. Fast nonsymmetric iterations and preconditioning for Navier-Stokes equations. *SIAM J. Sci. Comput.*, 17(1):33–46, 1996.

[27] H. C. Elman, D. J. Silvester, and A. J. Wathen. Performance and analysis of saddle point preconditioners for the discrete steady-state Navier-Stokes equations. *Numer. Math.*, 90(4):665–688, 2002.

[28] H. C. Elman, D. J. Silvester, and A. J. Wathen. *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*. Numerical Mathematics and Scientific Computation. Oxford University Press, New York, 2005.

[29] L. F. García. Circulación termohalina. Wikipedia.

[30] D. K. Gartling and C. R. Dohrmann. Quadratic finite elements and incompressible viscous flows. *Comput. Methods Appl. Mech. Engrg.*, 195(13-16):1692–1708, 2006.

[31] P. R. Gent, J. Willebrand, T. J. McDougall, and J. C. McWilliams. Parameterizing eddy-induced tracer transports in ocean circulation models. *J. Phys. Oceanogr.*, 25:463–474, 1995.

[32] A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, 10:345–363, 1973.

[33] A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall Inc., Englewood Cliffs, N.J., 1981. Prentice-Hall Series in Computational Mathematics.

[34] A. George and J. W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Rev.*, 31(1):1–19, 1989.

[35] R. Glowinski and P. Le Tallec. *Augmented Lagrangian and operator-splitting methods in nonlinear mechanics*, volume 9 of *SIAM Studies in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1989.

[36] S. M. Griffies. *Fundamentals of ocean-climate models*. Princeton University Press, Princeton, USA, 2004.

[37] I. Gruais, N. M. M. Cousin-Rittemard, and H. A. Dijkstra. A priori estimations of a global homotopy residue continuation method. *Numer. Funct. Anal. Optim.*, 26(4-5):507–521, 2005.

[38] M. Hagemann and O. Schenk. Weighted matchings for preconditioning symmetric indefinite linear systems. *SIAM J. Sci. Comput.*, 28(2):403–420 (electronic), 2006.

[39] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations. I Non-stiff Problems*, volume 8 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 2nd edition, 1993.

[40] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R .P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the Trilinos Project. *ACM Trans. Math. Software*, 31(3):397–423, 2005.

[41] A. von der Heydt and H. A. Dijkstra. Localization of multidecadal variability: I. Cross equatorial transport and interbasin exchange. *J. Phys. Oceanogr.*, 2007. In press.

[42] R. L. Higdon and R. A. de Szoeke. Barotropic-baroclinic time splitting for ocean circulation modeling. *J. Comput. Phys.*, 135(1):30–53, 1997.

[43] N.J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2nd edition, 2002.

[44] R. X. Huang. Mixing and energetics of the oceanic thermohaline circulation. *J. Phys. Oceanogr.*, 29:727–746, 1999.

[45] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392 (electronic), 1998.

[46] D. Kay, D. Loghin, and A. J. Wathen. A preconditioner for the steady-state Navier-Stokes equations. *SIAM J. Sci. Comput.*, 24(1):237–256 (electronic), 2002.

[47] H. B. Keller. Numerical solution of bifurcation and nonlinear eigenvalue problems. In P. H. Rabinowitz, editor, *Applications of Bifurcation Theory*. Academic Press, New York, U.S.A., 1977.

[48] S. Levitus and T.P. Boyer. World Ocean Atlas 1994, Volume 4: Temperature. *NOAA Atlas NESDIS 4, US Department of Commerce, Washington DC*, pages 0–117, 1994.

[49] S. Levitus, R. Burgett, and T.P. Boyer. World Ocean Atlas 1994, Volume 3: Salinity. *NOAA Atlas NESDIS 3, US Department of Commerce, Washington DC*, pages 0–99, 1994.

[50] C. Li and C. Vuik. Some results on the eigenvalue analysis of a SIMPLER preconditioned matrix. Report 03-08, Delft University of Technology, Department of Applied Mathematical Analysis, Delft, 2003.

[51] C. Li and C. Vuik. Eigenvalue analysis of the SIMPLE preconditioning for incompressible flow. *Numer. Linear Algebra Appl.*, 11(5-6):511–523, 2004.

[52] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11(1):134–172, 1990.

[53] G. Meurant. *Computer Solution of Large Linear Systems*. North-Holland Publishing Co., Amsterdam, 1999.

[54] A. C. de Niet and F. W. Wubs. Two preconditioners for the saddle point equation. In *Proceedings of the European Congres on Computational Methods in Applied Sciences and Engineering (ECCOMAS, Jyväskylä, 2004)*. See http://www.mit.jyu.fi/eccomas2004/.

[55] A. C. de Niet and F. W. Wubs. Two saddle point preconditioners for fluid flows. *Int. J. Numer. Methods Fluids*, 54(4):355–377, 2007.

[56] A. C. de Niet, F. W. Wubs, A. Terwisscha van Scheltinga, and H. A. Dijkstra. A tailored solver for bifurcation analysis of ocean-climate models. *J. Comput.*, 2007. Submitted.

[57] M. A. Olshanskii. A low order Galerkin finite element method for the Navier-Stokes equations of steady incompressible flow: a stabilization issue and iterative methods. *Comput. Methods Appl. Mech. Engrg.*, 191(47-48):5515–5536, 2002.

[58] M. A. Olshanskii and A. Reusken. Grad-div stabilization for Stokes equations. *Math. Comp.*, 73(248):1699–1718 (electronic), 2004.

[59] Intergovernmental Panel on Climate Change. Climate change 2007: The physical science basis, summary for policymakers. Fourth Assessment Report, February 2007. See www.ipcc.ch.

[60] S. V. Patankar. *Numerical Heat Transfer and Fluid Flow*. McGraw-Hill, New York, 1980.

[61] S. Röllin and O. Schenk. Maximum-weighted matching strategies and the application to symmetric indefinite systems. In *Lecture Notes in Computer Science LNCS 3732*, pages 808–817. 2005.

[62] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, 1993.

[63] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2003.

[64] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7(3):856–869, 1986.

[65] Y. Saad and H. A. van der Vorst. Iterative solution of linear systems in the 20th century. *J. Comput. Appl. Math.*, 123(1-2):1–33, 2000. Numerical analysis 2000, Vol. III. Linear algebra.

[66] O. Schenk and K. Gärtner. Solving unsymmetric sparse sytems of linear equations with pardiso. *Futur. Gener. Comp. Syst.*, 20(3):475–487, 2004.

[67] O. Schenk and K. Gärtner. On fast factorization pivoting methods for symmetric indefinite systems. *Elec. Trans. Numer. Anal.*, 23:158–179, 2006.

[68] O. Schenk, K. Gärtner, and W. Fichtner. Efficient sparse *LU* factorization with left-right looking strategy on shared memory multiprocessors. *BIT*, 40(1):158–176, 2000.

[69] R. Seydel. *Practical Bifurcation and Stability Analysis, From Eequilibrium to Chaos*. Springer-Verlag, New York, 2nd edition, 1994.

[70] V. E. Shamanskii. A modification of Newton's method. *Ukrain. Math. J.*, 19:118–122, 1967.

[71] D. J. Silvester and A. J. Wathen. Fast iterative solution of stabilised Stokes systems. II. Using general block preconditioners. *SIAM J. Numer. Anal.*, 31(5):1352–1367, 1994.

[72] G. L. G. Sleijpen and H. A. Van der Vorst. A Jacobi-Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 17(2):401–425, 1996.

[73] G. L. G. Sleijpen and F. W. Wubs. Exploiting multilevel preconditioning techniques in eigenvalue computations. *SIAM J. Sci. Comput.*, 25(4):1249–1272, 2003.

[74] L. A. Te Raa and H. A. Dijkstra. Instability of the thermohaline ocean circulation on interdecadal time scales. *J. Phys. Oceanogr.*, 32:138–160, 2002.

[75] A. D. Terwisscha van Scheltinga. *Data-assimiliation using implicit models*. PhD thesis, University of Utrecht, 2007. To appear.

[76] A. D. Terwisscha van Scheltinga and H. A. Dijkstra. Nonlinear data-assimilation using implicit models. *Nonlinear Processes in Geophysics*, 12(4):515–525, 2005.

[77] S. A. Thorpe. *The Turbulent Ocean*. Cambridge University Press, UK, 437 pp., 2005.

[78] G. Tiesinga, F. W. Wubs, and A. E. P. Veldman. Bifurcation analysis of incompressible flow in a driven cavity by the Newton-Picard method. *J. Comput. Appl. Math.*, 140(1-2):751–772, 2002.

[79] K. E. Trenberth, J. G. Olson, and W. G. Large. A global ocean wind stress climatology based on ECMWF analyses. Technical report, National Center for Atmospheric Research, Boulder, CO, U.S.A., 1989.

[80] M. Tůma. A note on the LDL^T decomposition of matrices from saddle-point problems. *SIAM J. Matrix Anal. Appl.*, 23(4):903–915 (electronic), 2002.

[81] H. A. van der Vorst. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13(2):631–644, 1992.

[82] H. A. van der Vorst. *Iterative Krylov Methods For Large Linear Systems*, volume 13 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2003.

[83] R. J. Vanderbei. Symmetric quasidefinite matrices. *SIAM J. Optim.*, 5(1):100–113, 1995.

[84] S. A. Vavasis. Stable numerical algorithms for equilibrium systems. *SIAM J. Matrix Anal. Appl.*, 15(4):1108–1131, 1994.

[85] C. Vuik and A. Saghir. The Krylov accelerated SIMPLE(R) method for incompressible flow. Report 02-01, Delft University of Technology, Department of Applied Mathematical Analysis, Delft, 2002.

[86] W. Weijer, H. A. Dijkstra, H. Öksüzoğlu, F. W. Wubs, and A. C. de Niet. A fully-implicit model of the global ocean circulation. *J. Comput. Phys.*, 192:452–470, 2003.

[87] F. W. Wubs, A. C. de Niet, and H. A. Dijkstra. The performance of implicit ocean models on B- and C-grids. *J. Comput. Phys.*, 211:210–228, 2006.

[88] C. Wunsch and R. Ferrari. Vertical mixing, energy and the general circulation of the oceans. *Annual Review of Fluid Mechanics*, 36:281–314, 2004.

[89] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Algebraic Discrete Methods*, 2(1):77–79, 1981.

[90] D. M. Young. *Iterative solution of large linear systems*. Academic Press, New York, 1971.

Samenvatting

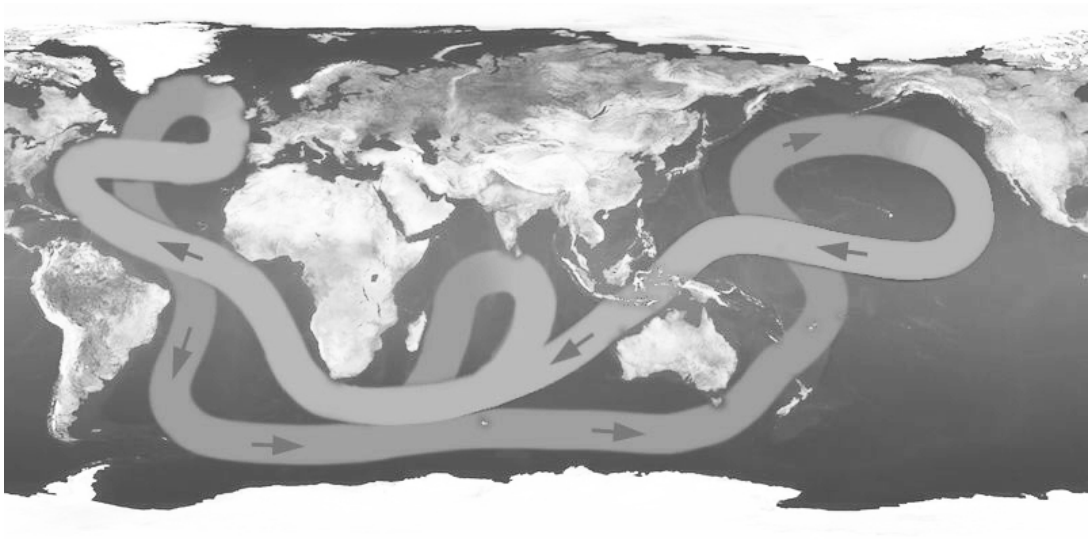
Oceaanstroming en klimaatverandering

In februari 2007 bracht het Intergovernmental Panel on Climate Change (IPCC) een rapport [59] uit met daarin de meest recente wetenschappelijke gegevens rond het broeikaseffect. De boodschap was duidelijk: de opwarming van de aarde staat buiten kijf en het is zeer waarschijnlijk dat de mens er een groot aandeel in heeft. De stijging van de gemiddelde temperatuur heeft tot gevolg dat de ijskappen smelten. Wetenschappers voorspellen dat de Noordpool binnen enkele decennia in de zomer volledig ijsvrij zal zijn. De impact van het verdwijnen van het drijvende ijs op de Noordpool is echter beperkt. Veel dramatischer is het als het landijs op Groenland gaat smelten. De enorme hoeveelheid zoet water die daarbij vrijkomt, kan zeer grote gevolgen hebben voor het klimaat op aarde.

Een voorbeeld is het milde klimaat in Noordwest-Europa. Dit wordt veroorzaakt door de Golfstroom die warm water vanuit de Golf van Mexico via de Noord-Atlantische oceaan naar Europa transporteert. Op weg naar het noorden koelt het water af, zinkt en vervolgt haar weg over de bodem in zuidelijke richting. Onder wetenschappers is een debat gaande over de stabiliteit van de Golfstroom. Kan de instroom van een grote hoeveelheid zoet water door smeltend ijs op Groenland de Golfstroom destabiliseren en tot stilstand brengen? Dat zou grote invloed hebben op het klimaat in Europa.

De Golfstroom is onderdeel van de grotere globale oceaanstroming die ook wel de ”conveyor belt”(transportband) wordt genoemd, zie Figuur 10.1. Het kenmerkende van de stroming is dat warm zout water in het noorden van de Atlantische oceaan afkoelt, zinkt, over de bodem naar de Indische en Grote of Stille oceaan stroomt, daar weer aan de oppervlakte komt, weer terug stroomt naar de Atlantische oceaan en onderweg als gevolg van de zon warmer en zouter wordt. Factoren die van invloed zijn op de stroming zijn: (i) de wind aan het oppervlak, (ii) verschillen in temperatuur als gevolg van de zon en (iii) verschillen in zout gehalte als gevolg van verdamping, neerslag, instroom van rivieren en smeltwater van de ijskappen. De stroming wordt verder bepaald door de draaiïng van de aarde, die een sterke Coriolis kracht geeft, en door de aanwezigheid van continenten en een behoorlijk ruige bodemtopografie.

Het klimaat op aarde wordt voor een belangrijk deel bepaald door de globale oceaanstroming. Het is dan ook van groot belang om te weten hoe de oceaan reageert op verstoring door bijvoorbeeld een grote hoeveelheid zoet water. De manier om daar achter te komen is het ontwikkelen van een computermodel dat de globale oceaanstroming als weergegeven in Figuur 10.1 kan berekenen. Daarvoor is het allereerst nodig alle natuurkundige processen die



Figuur 10.1: De globale oceaan circulatie (uit [29]).

een rol spelen, goed te vertalen naar een wiskundig model.

Daarnaast willen we gebruik maken van satellietgegevens over de waterhoogte en de geoïde om de stroming aan het oppervlak te bepalen. De geoïde is een oppervlak rond de aarde waarvoor de zwaartekracht in elk punt loodrecht op dit oppervlak staat en dientengevolge is er in elk punt op dit oppervlak geen kracht evenwijdig aan het oppervlak. Deze geoïde is geen perfecte bol, omdat de aarde dat niet is. Bovendien beïnvloeden bergen en troggen lokaal de zwaartekracht. De GOCE missie die eind 2007 gelanceerd zal worden, gaat de geoïde bepalen met enkele centimeters nauwkeurigheid. Voor de waterhoogte zijn al nauwkeurige data beschikbaar.

Als er geen stroming zou zijn in de oceanen zou het wateroppervlak samenvallen met de geoïde. De aanwezigheid van stroming zorgt ervoor dat wateroppervlak en geoïde van elkaar afwijken. In het geval van de Golfstroom is die afwijking ongeveer een meter. Door het verschil tussen geoïde en waterhoogte over de hele aarde te bepalen krijgen we een indruk van de stroming aan het oppervlak. Dat stromingsprofiel gebruiken we vervolgens om het model te verbeteren.

Het berekenen van de globale oceaanstroming

Aan het Institute for Marine and Atmospheric research Utrecht (IMAU) is de afgelopen jaren het computermodel THCM ontwikkeld. Dit model berekent de volledige driedimensionale oceaanstroming, dat wil zeggen de stroomsnelheden in drie richtingen, de druk, de temperatuur en het zoutgehalte. Deze grootheden hangen af van de plaats en de tijd en zijn onderhevig aan natuurkundige wetten voor behoud van impuls (in drie richtingen), massa, warmte en zout. De wiskundige vergelijkingen die het resultaat zijn van deze behoudswetten staan uitgebreid beschreven in Hoofdstuk 2. Deze *niet-lineaire partiële differentiaalvergelijkingen* zijn te complex

om analytisch op te lossen, met andere woorden, het is onmogelijk om een algemene functie voor een grootheid als de temperatuur te bepalen, waarin we een willekeurig punt in tijd en ruimte kunnen invullen om daar de exacte temperatuur te krijgen. Noodgedwongen benaderen we de oplossing door een rooster over de oceaan te leggen. In elke cel van het rooster nemen we aan dat de grootheden constant zijn. De oorspronkelijke vergelijkingen kunnen worden vertaald naar benaderende vergelijkingen op het rooster, wat resulteert in een serie vergelijkingen die afhankelijkheden geven voor grootheden in cellen die bij elkaar in de buurt liggen. Door deze benadering is het aantal vergelijkingen niet langer zes, maar zes maal het aantal cellen in het rooster. Hoe fijner het rooster hoe meer detail we kunnen vangen in de oplossing en hoe nauwkeuriger het model is. De grootte van de cellen hangt dus direct samen met de nauwkeurigheid van de oplossing. Ook het model staat uitgebreid beschreven in Hoofdstuk 2.

Aan het begin van het project waarvan de resultaten in dit proefschrift staan beschreven, bestond het computer model THCM al. De mogelijkheden waren echter beperkt doordat een model-run erg veel rekentijd en computergeheugen vergde, met als gevolg dat we noodgedwongen een grove resolutie gebruikten. Om realistischer berekeningen te doen is een fijner rooster nodig. Het is noodzakelijk de rekentechnieken te verbeteren om een stap vooruit te kunnen doen in oceaan-klimaat modellering. De grote uitdaging voor dit proefschrift is het verbeteren van de rekenmethoden, zodat het mogelijk is op grote schaal te rekenen aan het oceaanmodel met voldoende nauwkeurigheid.

Het oplossen van een stelsel vergelijkingen

Het knelpunt in de berekeningen is het oplossen van de vergelijkingen. Voordat we uit kunnen leggen wat en hoe we dit probleem hebben aangepakt, is het nodig iets uit te leggen over het oplossen van stelsels vergelijkingen. Ik zal dat doen aan hand van een klein voorbeeld. Stel dat we (slechts) drie onbekenden hebben, die we de naam geven: x_1, x_2 en x_3. Voor deze drie onbekenden gelden de volgende drie vergelijkingen:

$$\begin{aligned} 2x_1 &+ 4x_2 - x_3 = 9, \\ -4x_1 &+ 2x_2 - 3x_3 = -3, \\ -x_1 &+ 3x_2 + 4x_3 = -3. \end{aligned} \tag{10.1}$$

Deze vergelijkingen zijn allemaal lineair, dat wil zeggen, de onbekenden komen alleen voor in combinatie met een constante factor. Kwadraten of andere ingewikkelde functies van x_1, x_2 en x_3 spelen geen rol. We hebben drie onbekenden en drie vergelijkingen, die in dit geval een unieke oplossing geven voor x_1, x_2 en x_3. Om dit stelsel op te lossen, schrijven we het in *matrix-vector* vorm

$$Ax = b, \tag{10.2}$$

waarbij A een matrix (een vierkant blok getallen) is, x de vector met onbekenden en b een vector met de waarden aan de rechterkant van het ongelijk teken. Voor ons voorbeeld betekent

het dat de matrix en de vectoren er als volgt uit zien:

$$A = \begin{bmatrix} 2 & 4 & -1 \\ -4 & 2 & -3 \\ -1 & 3 & 4 \end{bmatrix}, \; x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \; b = \begin{bmatrix} 9 \\ -3 \\ -3 \end{bmatrix}.$$

We krijgen de oorspronkelijke vergelijking terug door de matrix-vector vermenigvuldiging uit te schrijven. Elke rij van A vermenigvuldigen we me de vector x door element voor element te vermenigvuldigen en het resultaat te sommeren.
De klassieke manier om een dergelijke vergelijking op te lossen is het ontbinden van de matrix A in een product van twee eenvoudiger matrices

$$A = LU, \tag{10.3}$$

waarbij L een onderdriehoeksmatrix is (alleen maar nullen boven de diagonaal), en U een bovendriehoeksmatrix (enkel nullen onder de diagonaal). Ons voorbeeld geeft de factoren

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -0.5 & 0.5 & 1 \end{bmatrix} \text{ en } U = \begin{bmatrix} 2 & 4 & -1 \\ 0 & 10 & -5 \\ 0 & 0 & 6 \end{bmatrix}. \tag{10.4}$$

We kunnen nu x bepalen in twee stappen. We lossen eerst de vergelijking

$$Ly = b \tag{10.5}$$

op, gevolgd door de vergelijking

$$Ux = y. \tag{10.6}$$

Als we beide vergelijkingeng combineren om de onbekende vector y weg te werken, zien we dat we op deze manier de juiste oplossing bepalen voor x. Immers $b = Ly = L(Ux) = LUx = Ax$.

Als we de matrix-vector vermenigvuldiging in vergelijking (10.5) uitschrijven krijgen we het stelsel

$$\begin{aligned} y_1 &= 9, \\ -2y_1 + y_2 &= -3, \\ -0.5y_1 + 0.5y_2 + y_3 &= -3. \end{aligned}$$

Door de onderdriehoeksvorm van de matrix L is dit stelsel eenvoudig op te lossen. De eerste vergelijking geeft onmiddellijk $y_1 = 9$. Gegeven y_1 kunnen we met de tweede vergelijking y_2 bepalen: $y_2 = (-3 + 2y_1) = (-3 + 18) = 15$. Tenslotte geeft de derde vergelijking $y_3 = (-3 + 0.5y_1 - 0.5y_2) = (-3 + 4.5 - 7.5) = -6$.

Nu de vector y bekend is, kunnen we x oplossen door vergelijking (10.6) uit te schrijven:

$$\begin{aligned} 2x_1 + 4x_2 - x_3 &= 9, \\ 10x_2 - 5x_3 &= 15, \\ 6x_3 &= -6. \end{aligned}$$

Omdat de matrix U een bovendriekhoeksvorm heeft, beginnen we op te lossen vanaf de laatste vergelijking, waaruit onmiddellijk volgt $x_3 = -1$. Dat gebruiken we in de tweede vergelijking om $x_2 = 1$ te berekenen. Tenslotte geeft de eerste vergelijking $x_1 = 2$. Dus onze oplossing is

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix}.$$

Door deze uitkomst in te vullen in vergelijking (10.1) valt eenvoudig te controleren dat dit inderdaad de oplossing is. Deze aanpak, waarbij een LU-ontbinding wordt gemaakt, die vervolgens wordt gebruikt om de oplossing in twee stappen te berekenen, wordt ook wel *direct* genoemd.

De stelsels in het oceaanmodel zijn vele malen groter dan dit eenvoudige voorbeeld. In het model lossen we vergelijkingen op van de vorm (10.2), maar dan met een matrix A die honderdduizend of meer rijen en kolommen bevat. In onze toepassing is de matrix A bovendien erg *ijl*, wat wil zeggen dat de matrix voor het grootste deel uit nullen bestaat. Per rij zijn er hoogstens 10 elementen ongelijk aan nul. We kunnen van de ijle structuur gebruik maken, door alleen de niet-nullen te onthouden. Dat scheelt erg veel computer geheugen. Echter bij het bepalen van de LU-ontbinding zoals in (10.3), neemt het aantal niet-nullen in de factoren L en U snel toe. We kunnen dat enigszins voorkomen door de volgorde van de onbekenden in x aan te passen. Voor de matrix A betekent dat een symmetrische verwisseling van rijen en kolommen. Zelfs als we dat erg goed doen zal er voor de opslag van de factoren L en U aanzienlijk meer geheugen nodig zijn dan voor de opslag van A. Bovendien neemt het geheugengebruik harder toe naar mate het probleem groter wordt. Het aanlopen tegen de grenzen van het beschikbare geheugen is een typische beperking op de toepasbaarheid van directe methoden.

Als alternatief voor de directe aanpak kunnen we een *iteratieve methode* gebruiken. Daarbij berekenen we de oplossing niet rechtstreeks, maar we construeren een rij $\{x^{(0)}, x^{(1)}, \ldots, x^{(n)}\}$ van steeds betere benaderingen voor die oplossing. In het algemeen gebruiken iteratieve methoden veel minder geheugen. Het nadeel is echter dat er in veel gevallen erg veel iteraties nodig zijn voordat er een benadering wordt gevonden die dicht genoeg bij de oplossing ligt. We kunnen de prestaties van iteratieve methoden flink verbeteren door *preconditionering*. De essentie van preconditionering is dat we niet de vergelijking (10.2) oplossen maar een vergelijking

$$P\tilde{x} = b. \tag{10.7}$$

waarbij P een matrix is, die gekozen is zodat

(i) P een goede benadering is voor A,

(ii) P eenvoudig te bouwen is,

(iii) een vergelijking met P makkelijker op te lossen is dan een vergelijking met A.

Omdat we in (10.7) een andere vergelijking oplossen dan in (10.2), zijn de oplossingen x en $\tilde{x}$ niet aan elkaar gelijk. Echter, als we een goede preconditioner kiezen, zal het verschil $x-\tilde{x}$ niet al te groot zijn en dus zal $b-A\tilde{x}$ dichtbij nul liggen. De correctie Δx die nodig is om $\tilde{x}$ naar de exacte oplossing te brengen volgt uit $A\Delta x = b - A\tilde{x}$. In deze vergelijking vervangen we de eerste matrix A opnieuw door de preconditioner; dit geeft $P\Delta x = b - A\tilde{x}$. Met de oplossing voor Δx volgt nu een nieuwe betere benadering voor de exacte oplossing: $\hat{x} = \tilde{x} + \Delta x$. Als deze nieuwe benadering niet goed genoeg is herhalen we deze procedure. In het algemeen geldt: hoe beter de preconditioner hoe sneller de convergentie in een iteratieve methode.

Een voorbeeld van zo'n preconditioner is een niet-exacte LU ontbinding. In plaats van (10.3) berekenen we de ontbinding

$$A = \tilde{L}\tilde{U} + E, \tag{10.8}$$

waarbij $\tilde{L}$ en $\tilde{U}$ ijlere benaderingen zijn voor de exacte factoren L en U. Doordat de factoren niet exact zijn, maken we een fout die we in de matrix E terug vinden. Doordat $\tilde{L}$ en $\tilde{U}$ veel ijler zijn, is het allereerst eenvoudiger om ze te bouwen. Daarnaast is er minder geheugen nodig om de matrices op te slaan en het oplossen van een stelsel vergelijkingen met deze boven- en onderdriehoeksmatrix is ook nog eens een stuk goedkoper. Het nadeel van iteratieve methoden is dat het niet eenvoudig is om een goede preconditioner te vinden voor complexe problemen zoals ons oceaan model.

In Hoofdstuk 3 worden directe en iteratieve methoden voor het oplossen van een stelsel vergelijkingen uitgebreider beschreven.

Het oplossen van vergelijkingen uit het oceaanmodel

De focus van dit proefschrift ligt in het ontwikkelen van nieuwe preconditioners voor de matrices uit het oceaanmodel. In de oorspronkelijk versie van het model, werd voor het oplossen van stelsels vergelijkingen gebruik gemaakt van MRILU een inexacte LU ontbinding. Deze methode werkt erg goed op een hele reeks van matrices, maar helaas heeft MRILU moeite met de matrices uit het oceaanmodel. Er is te veel geheugen en tijd nodig om een preconditioner met voldoende kwaliteit te bouwen.

De oorzaak van de moeiten die solvers in het algemeen hebben met de oceaanmatrices ligt voor een belangrijk deel in het feit dat onze problemen een aantal grote nul-blokken op de diagonaal bevat. Voor de structuur zie de inleiding van Hoofdstuk 7. De lege diagonaal blokken worden veroorzaakt door de onderliggende natuurkundige vergelijkingen en ze geven problemen bij het construeren van de (inexacte) LU ontbindingen in vergelijking (10.3) en (10.8). Soortgelijke problemen komen voor bij een deelprobleem van de oceaan matrix: het *zadelpuntprobleem*, waarbij we een vergelijking oplossen met een matrix van de vorm

$$\begin{bmatrix} A & B \\ B^T & 0 \end{bmatrix}, \tag{10.9}$$

hierin zijn A en B grote ijle matrices, B^T is de getransponeerde van B, dat is, de kolommen van B zijn de rijen van B^T. Zadelpuntproblemen komen veel voor als het resultaat van een stromingsvergelijking. Kennis van het oplossen van zadelpuntproblemen kan ons helpen bij het oplossen van vergelijkingen uit het oceaanmodel. Daarom staan zadelpuntproblemen in een groot deel van het proefschrift centraal.

Een combinatie van directe en iteratieve methoden

Het oorspronkelijke idee voor het ontwikkelen van een betere solver voor de oceaanmatrices was het integreren van directe en iteratieve methoden. Door de preconditioner MRILU aan te passen met gebruik van technieken uit directe methoden, zouden we de kwaliteit van de ontbinding kunnen verbeteren, waarbij minder geheugen en minder rekentijd nodig zouden zijn. In Hoofdstuk 5 hebben we de mogelijkheden van zo'n combinatie onderzocht. Voor de eenvoudiger Poisson vergelijking gaat dat prima, maar zodra we de stap maken naar het zadelpuntprobleem, worden de prestaties aanzienlijk slechter. De conclusie uit dat hoofdstuk is dat het onwaarschijnlijk is dat de incorporatie van dergelijk technieken in MRILU ons werkelijk veel verder zullen brengen.

Een directe methode voor het zadelpuntprobleem

In Hoofdstuk 4 ontwerpen we een nieuwe directe methode voor zadelpuntproblemen zoals in (10.9) met wat extra beperkingen op de structuur van deelmatrix B. Het algoritme heeft een aantal aantrekkelijke eigenschappen. Zo blijft bijvoorbeeld veel van de structuur van de oorspronkelijke matrix behouden tijdens het ontbinden van de matrix in factoren L en U. Een belangrijk resultaat uit dit hoofdstuk is een nieuw bewijs dat de ontbinding gegarandeerd numeriek stabiel is met een kleine groeifactor. Met andere woorden: het is gegarandeerd dat de ontbinding bestaat en dat de grootte van de elementen in L en U begrensd zijn. Dat betekent dat de oplossing x die we berekenen met behulp van de factoren, voldoende nauwkeurig is.

Numerieke experimenten laten zien dat onze aanpak ijlere factoren L en U oplevert dan bestaande methoden.

Pogingen om deze directe methode om te zetten in een preconditioner gecombineerd met een iteratieve methode zijn vooralsnog op niets uitgelopen. Kleine aanpassingen in de richting van een inexacte ontbinding hadden tot gevolg dat het aantal iteraties in de iteratieve methode snel opliep.

Preconditionering voor het zadelpuntprobleem

Voor zadelpuntproblemen valt het niet mee om inexacte en exacte ontbindingen te combineren. Dat wordt voor een deel veroorzaakt door het feit dat MRILU en veel andere solvers de matrix als een zwarte doos beschouwen. Er wordt geen rekening gehouden met de herkomst van de matrix. Alle onbekenden en vergelijkingen worden gelijk behandeld. Juist bij zadelpuntproblemen zijn er twee duidelijk verschillende grootheden: stroomsnelheid en druk, waarbij de

laatste grootheid het nul blok in (10.9) veroorzaakt. Een aanpak die de verschillen tussen de grootheden en de structuur in de matrix respecteert blijkt zeer succesvol.

In Hoofdstuk 6 vergelijken we een aantal preconditioners uit de literatuur (o.a. SIMPLE(R), en preconditioners van Wathen, Elman en Silvester) met twee nieuwe preconditioners, die geinspireerd zijn door tijdsafhankelijke methoden. Een preconditioner gebruikt grad-div stabilisatie, de andere artificiële compressibiliteit. In alle gevallen gaat het om preconditioners die de snelheden anders behandelen dan de druk. Via eigenwaarden analyses en numerieke experimenten laten we zien dat de nieuwe preconditioners competitief zijn op uiteenlopende zadelpuntproblemen, waaronder een zadelpuntprobleem uit de oceaanmatrix.

Preconditionering in het oceaan model

Geinspireerd door het succes van de structurele aanpak bij zadelpuntproblemen, hebben we onderzocht of we die aanpak ook op de oceaanmatrix kunnen toepassen. In Hoofdstuk 7 beschrijven we de nieuwe preconditioner die het resultaat is van dat onderzoek. In het oceaanmodel spelen zes verschillende grootheden een rol. We kunnen ze nog wel enigszins groeperen. De stroomsnelheden in het horizontale vlak hebben veel overeenkomsten. Hetzelfde geldt voor de temperatuur en het zoutgehalte. Deze twee types grootheden verschillen echter wezenlijk van de stroomsnelheid in de diepte en de druk. Die vier groepen grootheden worden na elkaar opgelost. De deelmatrices die we moeten oplossen in plaats van de grote oceaanmatrix, zijn eenvoudiger en goed te behandelen voor MRILU. De nieuwe preconditioner is goedkoper te bouwen, eenvoudiger toe te passen en zuiniger in het geheugengebruik.

In Hoofdstuk 8 laten we de prestaties van de nieuwe preconditioner zien op een aantal serieuze oceaanproblemen. De conclusie is eenduidig: dankzij de nieuwe preconditioner kunnen we de stromingen sneller berekenen, en ook nog eens op een fijner rooster, waardoor de nauwkeurigheid van de berekende oplossing veel beter is. Hoewel het moeilijk is om precies in een getal aan te geven wat de winst is, kunnen we met zekerheid zeggen dat we een orde van grootte sneller kunnen rekenen dan met de oude code. Waarmee het hoofddoel van dit onderzoek is bereikt.

Conclusie

Terug naar het begin van deze samenvatting, waarin het berekenen van de stabiliteit van de globale oceaanstroming centraal stond. Het nieuwe algoritme voor het oplossen van de vergelijkingen uit het oceaanmodel, zoals met name beschreven in de Hoofdstukken 7 en 8, is een grote stap vooruit. Doordat vergelijkingen veel sneller kunnen worden opgelost, kunnen we de grootschalige berekeningen doen, die nodig zijn om beter te kunnen voorspellen hoe de oceaan zal reageren op verstoringen.

Dankwoord

Nu dit proefschrift bijna af is, wordt het tijd een aantal mensen te bedanken die hebben bijgedragen aan het tot stand komen ervan.

De grootste dank gaat uit naar Fred Wubs. Ik ben me ervan bewust dat ik het goed heb getroffen met jou als dagelijks begeleider. Ik wil je bedanken voor de prettige samenwerking in de afgelopen vijf jaar. Ik heb het erg gewaardeerd dat je me veel ruimte hebt gegeven om mijn eigen weg te gaan. Ik mocht mijn eigen interesses en wiskundige intuïtie volgen. Je geduld en brede kennis is van veel waarde geweest op momenten dat het qua onderzoek wat minder ging. Gelukkig stond de deur letterlijk en figuurlijk altijd op een kier om met een probleem naar binnen te vallen. Het feit dat wij het geloof in dezelfde almachtige God deelden, heeft voor mij ook veel betekend. Ik hoop dat we ook in de toekomst regelmatig contact zullen houden.

Mijn dank gaat ook uit naar mijn promotor Arthur Veldman. Je was wat meer op afstand betrokken, maar altijd geïnteresseerd in het onderzoek. Je hebt de eerdere versies van dit proefschrift kritisch gelezen en ik heb de gesprekken over de inhoud naar aanleiding van je commentaar erg gewaardeerd. Ik wil je ook bedanken voor je steun in mijn strijd voor verlenging in verband met ouderschapsverlof.

Dit proefschrift was er ook niet geweest zonder Henk Dijkstra. Bedankt voor de prettige samenwerking in de afgelopen jaren. Je onuitputtelijke optimisme en aanstekelijke enthousiasme waren erg motiverend. Ik wil ook je vrouw Lia bedanken voor jullie gastvrijheid en de tripjes tijdens mijn verblijf in Fort Collins.

In het STW project was ik niet als enige AiO werkzaam. Arjen Terwisscha van Scheltinga, mijn collega in Utrecht, bedank ik voor de goede samenwerking. Je was een aangename huisgenoot in Fort Collins. Succes met het afronden en verdedigen van jouw proefschrift later dit jaar.

Ik wil de leden van de beoordelingscommissie, de hoogleraren Henk van der Vorst en Guus Stelling bedanken voor het kritisch lezen en beoordelen van mijn proefschrift.

I would like to thank Andy Salinger and Eric Phipps for their hospitality and help during my stay at Sandia National Laboratories. I really enjoyed the Albuquerque Isotopes, the climate and the food in New Mexico.

In de afgelopen jaren had ik het genoegen mijn werkkamer te delen met Geert Fekken, Dirk-Jan Kort, Rik Wemmenhove en Jonas Thies, die ik allemaal wil bedanken voor de koffie (van wisselende kwaliteit), thee en de aanspraak over wetenschappelijke hoogte- en dieptepunten.

Amid, Barteld, Erwin, Ena, Gerk, Gert-Jan, Joop, Joost, Lenny, Mark, Remke, Sunayana, Theresa bedankt voor de lunches en thee pauzes, die gelukkig zelden over wiskunde gingen.

Ik ben het instituut dankbaar dat ik onderwijs mocht geven. Ik heb het altijd als een grote uitdaging gezien om de wiskunde op begrijpelijke wijze over te dragen aan studenten. Als dat lukt geeft dat erg veel voldoening. Ik wil Henk Broer, Fieke Geurts, Jan van Maanen, Keimpe Nevenzeel, Marius van der Put en Erik Thomas bedanken voor de prettige samenwerking bij verschillende vakken.

Het secretariaat wil ik bedanken voor alle praktische ondersteuning.

Dit proefschrift was er zeker niet gekomen zonder een goede en krachtige computer met de juiste werkende software. Daarvoor heb ik regelmatig een beroep moeten doen op Harm, Jurjen en Peter, de systeembeheerders, die mijn problemen vaak sneller wisten op te lossen dan dat ik mijn mail met de probleembeschrijving kon typen. Mijn dank is groot.

Door de komst van mijn lieve dochters Marloes en Fieke is dit proefschrift later klaar dan aanvankelijk de planning was. Mijn wekelijkse ouderschapsverlofdag was echter een zeer aangename dag in de week. Juist omdat het onmogelijk was ook maar een pagina van een artikel te lezen.

De laatste regels in die proefschrift zijn voor Geertje. Ik denk dat jij uiteindelijk meer offers voor dit proefschrift hebt gebracht dan ik. Tijdens mijn verblijf in Fort Collins kreeg Marloes de waterpokken en Fieke was nog maar een paar maanden oud toen ik een maand in Albuquerque verbleef. Je stond er alleen voor in die zware periodes en toch liet je me gaan. Ook de eerste maanden van dit jaar waren niet zo gemakkelijk doordat ik wat prikkelbaar was vanwege de druk om het proefschrift af te maken. Ik ben je dankbaar voor alle steun en ruimte die je aan me hebt gegeven de afgelopen jaren. Je bent onmisbaar voor me.